Autonomous and Semi-Autonomous Control of Multi-Agent Task Allocation Systems

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

In Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

David Robert Schneider

January 2008

Autonomous and Semi-Autonomous Control of

Multi-Agent Task Allocation Systems

David Robert Schneider, Ph.D.

Cornell University 2008

The ever increasing need to solve a wide variety of NP-Hard task allocation problems effectively and in computation times acceptable for real-time application has motivated the development of the new optimal and large scale approximation methods presented in this dissertation. The chief new optimal method presented, referred to as G^*_{TA}, utilizes a minimum spanning forest algorithm to generate optimistic predictive costs within an A^* framework, and a greedy approximation method to create upper-bound estimates. The G^*_{TA} method is shown to run on average two orders of magnitude faster than a traditional Mixed Integer Linear Programming (MILP) technique, and a combined approach of G^*_{TA} and MILP, referred to as G^*MILP, is also presented for its scaling potential. The G^*_{TA} method is then improved further through the development of a new optimistic predictive cost function which leads to computational runtimes that are five times faster still, while using up to an order of magnitude less memory.

The G^*_{TA} method is also presented as an any-time solution capable approximation method, which then motivates the development of a G^*_{TA} based hierarchical approximation method called H-G^*_{TA}. The H-G^*_{TA} method is built upon and validates the premise that since these task allocation problems are NP-Hard, faster solutions to large scale problems can be obtained by partitioning these large problems into a

hierarchy of smaller sub-problems that can be solved within a reasonable amount of time. Details are provided on the integration within H-G^*_{TA} of a newly developed adaptive K-means partitioning technique, incorporation of G^*_{TA} to solve sub-problems optimally, combining the sub-problem solutions to obtain a final solution to the original problem, and on applying a K-Opt post-optimization technique.

As task allocation research is commonly associated with the applied field of robotics, this dissertation also presents a description of the Cornell RoboFlag system; a high fidelity, robotic testbed and simulation environment that was utilized for validating the computational runtime and the solution quality of all of the task allocation methods discussed. Finally, this dissertation presents a chapter on the educational robotics research that was carried out within the NASA Robotics Alliance independently of the task allocation research.

BIOGRAPHICAL SKETCH

David Schneider graduated from Rensselaer Polytechnic Institute (RPI) 1st in his class with a bachelor's degree in Chemical Engineering with a concentration in Electronic Media Arts and Communication in December, 1999. Just before coming to RPI David was recognized as a National Science Scholar by President Bill Clinton. During his time at RPI, some of the highest honors David received were the R.P.I. Founders Award of Excellence and R.P.I. Lewis S. Coonley Prize for Design, the latter of which is one of the two highest awards given to a Chemical Engineering RPI undergraduate. During his time at RPI, David also worked as a Chemical and a Mechanical Engineer for Proctor and Gamble and as Production Assistant and Screenwriter for Walt Disney Attractions Television Production.

After receiving his bachelors, David went onto Columbia University to pursue a Masters of Fine Arts in Film, concentrating on Screenwriting and Directing. While studying Film at Columbia University, David also became a full time instructor for the engineering core curriculum course, "Introduction to Engineering Design". In this role, David not only became a finalist for the 2001 Columbia Presidential Teaching Awards, and a Guest Speaker at the 2001 National SWE Conference, but much of his educational work was featured as one of three major projects for a $30,000,000 National Science Foundation, Columbia University proposal.

From this success, David left the Film M.F.A. program to pursue a Mechanical Engineering, Dynamics and Controls PhD at Cornell University as a step towards developing a career within academia. Within his first six months, David became the Program Manager for the Cornell / DARPA / Wright Patterson ARFL RoboFlag

project and within his first year, David received Best Presentation at the 2003 American Controls Conference first Interactive Session for the presentation of RoboFlag v2.1. From his work on the DARPA MICA project, David began to focus his research in the area of task allocation, which with the creation of such real-time optimal algorithms as the G^*_{TA} method, became the center point of his PhD research.

This research also earned David the opportunity to become a Team Lead in the very first year of the NASA Robotics Academy. David was stationed within NASA Goddard Code 588, the Advanced Architectures and Automation branch, where by the end of his contract, his team had won every honor awarded including 1st Place in the NASA Goddard Robotics Internship Program Overall Project Achievement category. Out of this work, David was also given the opportunity to co-found with Mark Leon, NASA AMES Director of Education, the NASA Robotics Alliance Cadets program; an educational programmed aimed at re-inventing the first two years of undergraduate Mechanical Engineering, Electrical Engineering, and Computer Science as a highly integrated and iterative NASA National Robotics Curriculum.

In addition to several task allocation and educational publications that David has earned along the way, David has most recently been recognized as a leading educator on a global scale as being one of eighteen researchers world-wide to receive an invitation to submit to a special edition of the International Journal of Engineering Education. His paper, "Active Learning and Assessment within the NASA Robotics Alliance Cadets Program" was recently accepted. David is now determined to complete his PhD at Cornell University and thereafter dedicated to continue his research in task allocation, robotics, and engineering education.

To My Mother and Father who have always encouraged me, supported me, and taught me to be my very best in everything that I do

To my sister for teaching me patience, to always enjoy what you do, where to place commas, and an uncanny ability to explain the unexplainable

To my compatriot in arms, Oliver "Spock" Purwin,
whom I could always count on, despite the odds

Most of all, to my fiancée, Anne… my God, I love you… for you have grown with me a love, partnership, and trusting friendship that I can always believe in.
I can't wait for our next adventure

ACKNOWLEDGMENTS

Mark Campbell

Hod Lipson

Mason Peck

Raffaello D'Andrea

Ephrahim Garcia

David Schwartz

Bart Selman

Carla Goomes

Don Greenburg

Stephen Hamilton

Matt Ulinski

Oliver Purwin

Andrey Klochko

Michael Babish

Jarurat Ousingsawat & Matt Earl

Ken Sterk & Jeremy Miller

Barbie Medina

Chris Edmonds & Avi Hoffman

Jeff Hosler

Mark Leon

David Lavery

Morton Friedman

Hugh O'Brian

2005-2007 NASA Robotics Alliance Cadets Teams

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF PSEUDO-CODE

LIST OF ABBREVIATIONS

In order of first appearance:

Abbreviation	Definition	Section
DARPA	Defense Advanced Research Projects Agency	1.1
AFRL	Air Force Research Laboratory	1.1
NASA	National Aeronautics and Space Administration	1.1
UAV	Unmanned Aircraft Vehicle	1.1
AUV	Autonomous Underwater Vehicle	1.1
NOAA	National Oceanic and Atmospheric Administration	1.1
GUI	Graphical User Interface	1.2
HITL	Human in the Loop	1.2
MILP	Mixed Integer Linear Programming	2
TSP	Traveling Salesman Problem	2.1
LP	Linear Programming	2.1
MPC	Model Predictive Control	2.1
ST-SR-TA	Single Task Robots – Single Robot Tasks – Time Extended Assignment	2.2
AEP	ALLIANCE Efficiency Problem	2.2
MST	Minimum Spanning Tree	2.4.3
MSF	Minimum Spanning Forest	2.4.3
MTSP	Multiple Salesman Traveling Salesman Problem	4.1
STEM	Science, Technology, Engineering, Mathematics	6.1
ABET	Accreditation Board for Engineering and Technology	6.1
ASEE	American Society of Engineering Education	6.1
ERM	Educational Research Methods	6.1
RCC	Robotics Curriculum Clearinghouse	6.2
CAT	Classroom Assessment Technique	6.3
FIRST	For Inspiration and Recognition of Science and Technology	6.4

LIST OF SYMBOLS

In order of first appearance:

Symbol	**Description**	**Section**
s_G	Global view sensor data for the RoboFlag testbed	1.3
s_L	Local view sensor data for the RoboFlag testbed	1.3
$\hat{s}_G$	Global view estimate for the RoboFlag testbed	1.3
s	Sensor data (local or global) for the RoboFlag testbed	1.3
pl	Information on plays for the RoboFlag testbed	1.3
d	Destination command for the RoboFlag testbed	1.3
vel_c	Velocity command for the RoboFlag testbed	1.3
Ru	Defined set of rules within RoboFlag	1.3.1
S_{obj}	Set of all objects that make up a RoboFlag scenario	1.3.1.1
Ty	Object type of a RoboFlag object	1.3.1.1
Tm	Team of an object within RoboFlag	1.3.1.1
P_S	Set of all RoboFlag scenario parameters	1.3.1.1
Ru_{Ty}	Set of rules specific for RoboFlag object type Ty	1.3.1.1
$ID\#$	RoboFlag object identification number	1.3.1.1
pos	Position of an object in RoboFlag	1.3.1.2
o	Orientation of an object in RoboFlag	1.3.1.2
vel	Velocity of an object in RoboFlag	1.3.1.2
ss_t	Scenario state of an object in RoboFlag	1.3.1.2
f	Current fuel level of an object in RoboFlag	1.3.1.2
r_{ss}	RoboFlag state change number	1.3.1.3
r_w	RoboFlag state weight	1.3.1.3
r_f	RoboFlag list of state functions	1.3.1.3
$ss_{t,i}$	Initial state of a RoboFlag object(prior to rule investigation)	1.3.1.3
$ss_{t,f}$	Final state of a RoboFlag object (after rule execution)	1.3.1.3
$r_{sum,ss}$	Sum of the RoboFlag state weights for an investigated rule	1.3.1.3
obj	Individual RoboFlag object	1.3.1.3
L	Set of all RoboFlag objects within a given local sensor range	1.3.1.4
$Bw_{i,j}$	maximum available bandwidth available for communication between RoboFlag Agent/HITL i and Agent/HITL j	1.3.2
$La_{i,j}$	The communication latency between RoboFlag Agent/HITL i and Agent/HITL j in frames.	1.3.2
Ch_k	RoboFlag messaging channel	1.3.2
Ms	Size of a RoboFlag message	1.3.2
F	Number of frames a RoboFlag message is communicated in	1.3.2
U_x	Low level input control parameter	1.3.5
U_y	Low level input control parameter	1.3.5

U_θ	Low level input control parameter	1.3.5
T_{delayOut}	Time delay added to outgoing message communications	1.3.5
n_a	Number of agents in a RoboFlag team	1.4
vel_a	The set of maximum velocities for all agents in a RoboFlag team	1.4
$\tau_{i,j}$	a test set collection of RoboFlag "games" where each game has i agents per team and each agent has a maximum velocity of j	1.4
$\%t_{MC}$	Percentage of time a human operator controlled agents manually in a RoboFlag scenario	1.4
$\%t_{AC}$	Percentage of time a human operator controlled agents using automation commands in a RoboFlag scenario	1.4.1
s	Number of sources	2.2
t	Number of targets	2.2
path	Means of transitioning for a source from its initial state to a target state or from one target state to another	2.2
trip	Ordered set of paths to be determined for each source, where Tr_i stands for the trip of source i	2.2
Tr_i	The trip of source i	2.2
J	Total cost of a task allocation solution	2.2
v	A single vertex	2.3.1
e_{ij}	Edge between vertex v_i and v_j	2.3.1
$w_{i,j}$	Weight of the edge between vertex v_i and v_j	2.3.1
E	Set of all edges	2.3.1
V	Set of all vertices	2.3.1
S	Set of all sources	2.3.1
T	Set of all targets	2.3.1
v	Number of vertices in set V	2.3.1
J^*	Cost of the optimal solution to a task allocation problem	2.3.1
n_{min}	Node in the set N with the minimum total estimated cost	2.4
R	Set of all resource usage history for a given node	2.4.1
R_{ij}	The resource usage history for resource j by source i	2.4.1
r	Number of resources	2.4.1
U	Set of all unassigned targets for a node	2.4.1
C_t	True cost of a node: cost required to execute just the partial solution of a node	2.4.1
C_p	optimistic predictive cost: optimistic estimation of the best remaining cost that would be incurred in completing a node's partial solution	2.4.1
$\hat{C}_f$	Final cost estimate: optimistic estimate of the best total cost that would could incurred from completing a node's partial solution	2.4.1
N	Set of all nodes sorted by ascending final cost estimates	2.4.1
C_f^*	The optimal final cost of a node: the best possible final cost	2.4.1

	that can be incurred from completing a node's partial solution	
setID	The temporary ID given to a vertex to designate which set it belongs to during the Minimum Spanning Forest creation	2.4.3
Q	List of set IDs used in the creation of the Minimum Spanning Forest	2.4.3
n	A single node	2.4.4
X	The set of all expanded nodes	2.4.4
$\hat{J}$	The upper bound cost for the task allocation solution	2.4.5
e_{min}	The smallest weight outgoing edge	2.4.5
v_{end}	The ending vertex of a trip	2.4.5
M	The primary matrix used in $MILP_{TA}$	2.5.1
b	The width of the M matrix, i.e. the sum of the number of possible permutations for the target set *T*, times *s*	2.5.1
C_v	The cost column vector used in $MILP_{TA}$, A^*MILP, and G^*MILP	2.5.1
A[]	Binary variable row vector used in $MILP_{TA}$, A^*MILP, and G^*MILP	2.5.1
M_{TSP}	The primary matrix used in A^*MILP, and G^*MILP	2.5.2
b_{TSP}	The width of the M_{TSP} matrix, the sum of the number of possible combinations of the target set *T*, times *s*	2.5.2
Pr_v	Set of the edges connected to vertex v that is also part of an associated MSF or MST, sorted by edge weight.	2.8.4
$C_{Tr,max}$	The highest or worst cost of any single trip within a node	2.8.7
C_{Tri}	The cost of source *i*'s trip within a node	2.8.7
E_{new}	Reduced edge set from the graphical representation of the task allocation problem's edge set, *E*, that has been simplified using the partial solution of a node	3.4.3
T_{new}	Target set for the simplified task allocation graphical representation E_{new}	3.4.3
S_{new}	Source set for the simplified task allocation graphical representation E_{new}	3.4.3
P	The set of all points to be clustered	4.2
p	The number of points to be clustered	4.2
$p_{c,max}$	The maximum number of points allowed in any one cluster	4.2
$k_{top,max}$	The maximum number of clusters allowed at the top level of the hierarchy	4.2
K	The set of all clusters	4.2
K_α	The set of all clusters at level α of the hierarchy	4.2
α	Arbitrary level of the cluster hierarchy	4.2
C_K	The set of all centroids for all clusters	4.2
k_0	The starting number of clusters in the initialization phase	4.3
P_c	Initial set of cluster centroids	4.3
k	A single cluster	4.3

P_k	The cluster members of cluster k, i.e. the set of points used to form cluster k	4.3
$D_{k,total}$	The sum of the distortion between all members of cluster k	4.3
D_k	The average distortion between all members of cluster k	4.3
$D_{max,k}$	The maximum distortion between any two members within cluster k	4.3
$D_{max,a}$	The maximum allowable distortion between any two members within the same cluster at level α	4.3
$J_{i,j}$	The cost between points *i* and *j*	4.3
K_{pair}	Set of cluster pairs	4.5.2
$D_{total,\alpha}$	The total distortion of all clusters at level α	4.6
$D_{total,max}$	The maximum allowable total distortion for all clusters at the same level	4.6.1
$D_{total,\Delta}$	The change in distortion between clustering iterations at the same level	4.6.1
K_{top}	The set of clusters at the top level of the hierarchy	4.7
$D_{max,0}$	The D_{max} value for the first level of the cluster hierarchy	4.7.4
$k_{top,add}$	The number of clusters that need to be made out of the top level of cluster so that the top level of clusters contains exactly $k_{top,max}$ clusters	4.8
$Tr_{i,top}$	The trip for source *i* at the top of the cluster hierarchy, i.e. a trip belonging to the solution of the hierarchical task allocation summary problem	5.3
$Tr_{i,final}$	The trip for source *i* that is part of the final solution to the original task allocation problem addressed by $H\text{-}G^*_{TA}$	5.3
β_i	Cost multiplying constant for source *i*	5.5.2
$\gamma_{i,j}$	Resource usage multiplying constant for source *i* and resource *j*	5.5.2
Ne(A,B)	The neighbor pair of clusters A and B	5.5.3
kopt	The value of "K" in the K-Opt algorithm	5.8
L_{seq}	Length of the total solution sequence being investigated in the K-Opt algorithm	5.8
T_α	Set of all targets being considered to be clustered at level α	5.11.1

PREFACE

Task allocation is a wide, ever growing area of research that covers a variety of problems where a series of tasks must be assigned to a group of agents that can carry out or complete those tasks. These problems are commonly associated in NP-Hard research problems that involve groups of robotic agents that must determine on their own how to effectively distribute the tasks so as to minimize the effort required to complete a given set of tasks. This PhD dissertation presents 5 pieces of innovative work which represent new contributions to this area of science.

The first chapter describes the testbed, Cornell's RoboFlag that was used in all of the task allocation research implementation tests presented in the chapters following. This chapter focuses on the latest version, Cornell's RoboFlag v.3.0, and represents the culmination of work performed by the author as the Program Manager of RoboFlag versions 1.1, 2.0, 2.1 and 3.0 under both Prof. Mark Campbell and Prof. Raffaello D'Andrea. This chapter also presents a discussion on how fully autonomous task allocation was effectively incorporated into a series of semi-autonomous task management experiments conducted in part with Scott Galster's group at Wright Patterson Air Force Research Laboratories.

With the testbed established, the second chapter introduces the main task allocation algorithm developed in this work, G^*_{TA}. This new algorithm is shown to produce guaranteed optimal solutions in computation times on average up to two orders of magnitude faster than a standard implementation of Mixed Integer Linear Programming, which was the leading task allocation method prior to this work. The G^*_{TA} method is described in terms of utilizing an A^* framework to provide the

guarantee of optimality while simultaneously incorporating a greedy method to provide upper bound estimates that effectively reduce the search space. This chapter concludes with a discussion of the G^*_{TA} method's ability to handle a diverse range of task allocation problem variations as well as the additional ability of the optimal G^*_{TA} method to be used very effectively as an approximation method under strict computation time constraints.

As a direct continuation of the second chapter's work, the third chapter presents an improved optimistic predictive cost function to replace G^*_{TA}'s original optimistic predictive cost function within the algorithm's A^* framework. This new optimistic predictive cost function is shown to provide more accurate estimations that lead to computation times that are on average up to 5 times faster than were possible using the original function, and to a reduction of up to an order of magnitude in the required average memory usage.

In the fourth chapter, a new clustering algorithm is presented which is a hierarchical adaptive version of the popular K-means algorithm. This clustering method is presented as a supporting algorithm to efficiently partition NP-Hard problems, like task allocation problems, into a hierarchy of sub-problems. From this work, Chapter 5 then presents a new hierarchical approximation variation of the G^*_{TA} method called H-G^*_{TA}, that allows very large scale task sets to be allocated in real-time. This new method is described in a five stage process of: 1) using the Chapter4 clustering method to create task cluster sub-problems, 2) characterizing the clusters in terms of a newly defined cluster property, neighbor pairs, and using this information to solve the sub-problems, 3) solving the task allocation summary problem at the top of the hierarchy, 4) recursively determine a solution to the original large scale task allocation problem

from the summary problem solution and 5) applying a post optimization technique to improve the final solution quality. The new H-G^*_{TA} method is also shown to produce solutions of comparable quality to a standard greedy method in average computation times up to an order or magnitude faster than the greedy standard.

Finally, as much of the work presented in this dissertation is implemented on robotic systems, Chapter 6, presents another area of robotics research that was carried out independently during the task allocation research. In Chapter 6, an introduction to the NASA Robotics Alliance Cadets Program is presented in the context of the incorporation of Active and Cooperative Learning with Assessment techniques as part of the creation of a highly innovative and integrated curriculum for the first two years of Mechanical Engineering, Electrical Engineering, and Computer Science. This chapter ends with a description of a Robotics Triathlon in-class competition that exemplifies the techniques and methodologies presented throughout the paper.

As this dissertation was written using the Cornell University PhD paper options standard for formatting, each chapter is written as a stand alone paper. Elements of these chapters may also have already been published in other journals, magazines and conferences, but the chapters presented here may have been modified slightly from their original publication in order to provided better continuity. However all elements presented here are the original work of the dissertation author.

CHAPTER 1

THE ROBOFLAG TEST SYSTEM FOR DECENTRALIZED AUTONOMOUS AND SEMI-AUTONOMOUS COOPERATIVE MULTI-AGENT RESEARCH

The need for a flexible platform to test the theory of cooperative multi-agent systems and bring these concepts into practice has become a common yet crucial challenge for many researchers. Over the past six years, Cornell University has been developing a testbed capable of both autonomous and semi-autonomous scenarios involving optional human operators to meet this need. These efforts have culminated in the development of the RoboFlag test system which is used extensively as a platform for research and education by a wide variety of institutions, including Cornell, Cal-Tech, the DARPA MICA Project, the Wright Patterson AFRL, the NASA Advanced Automations and Architectures Branch, and has been used as the main platform for this publication's extensive study of the fully-autonomous optimal and approximation task allocation problem.

This chapter discusses the capabilities of the latest release, version 3.0, of the largely open source RoboFlag system. Furthermore, as numerous studies with strong theoretical roots have been translated into applications via the RoboFlag platform, this chapter also offers an introduction to the RoboFlag system with the discussion of a study that addresses the opposite end of the cooperative multi-agent research spectrum; a study on operator decision modeling theory through a set of experiments involving human operators managing teams of semi-autonomous robots.

1.1 Motivation

The time and effort required to create a system to explore and validate the application potential of a theory is often one of the most challenging and resource draining aspects of multi-vehicle system cooperative control studies. Part of the reason for this trend is that many of the primary applications investigated are inherently expensive, ranging from coordinated UAV control [1.1],[1.2] to AUV coastal monitoring [1.3], and even to space exploration [1.4]. Out of this situation, the need arises to have an experimental platform that enables the level of interaction and challenges of these practically relevant problems to be replicated, from at least an algorithmic standpoint, in a test environment that is relatively low cost and low risk. Furthermore, the system must also be robust enough to ensure that the validity of the methods developed in the test system will be readily transferable to the intended end applications. The potential value of such a test system has been formally recognized by such organizations as the U.S. Department of Defense [1.5], the AFRL [1.6], and the NASA / NOAA Adaptive Sensor Fleet Project [1.7] to name only a few.

To address these needs, the RoboFlag system was originally conceived by Prof. Raffaello D'Andrea of Cornell as a combination of both a real robotic hardware and computer simulation test system [1.8],[1.9]. The first major test scenario was then developed in part with Prof. Richard Murray of Caltech [1.10]. This initial system provided the framework for a network of distributed robotic agents, including concepts for a rules system and an Arbiter program to enforce the scenario conditions. From this foundation, the initial system was then expanded, and more importantly robustness was added, in order to offer greater flexibility to the research community. This not only allowed a wider variety of studies to take advantage of the already proven fidelity of the system but also allowed different potential theories and methods

to be tested against each other within a common system. These studies ranged from cooperative estimation work, to path planning, to initial decision modeling, to simulating military scenarios [1.11]-[1.14]. RoboFlag then matured as both a hardware and software platform under the DARPA MICA project, under Cornell Profs. Mark Campbell and Raffaello D'Andrea and Prof. Murray at Cal-Tech. During this time, a series of over 350 tests were run which showed the RoboFlag simulator produced statistically similar results to the real robot hardware, [1.15]. The RoboFlag program has continued more recently with the release of the new RoboFlag v.3.0 system which offers, among other new features, a significant increase in the number of potential objects and teams within a game and in the number of parameters that are available to vary those objects and teams. Since its creation, a large amount of the RoboFlag test system has been available as open source code via [1.16] and the new RoboFlag v.3.0 system will continue this tradition.

This paper begins with a general overview of the RoboFlag system which is followed by a more technical breakdown of the most recent release's system architecture and key features. The paper then details two research concepts which were uniquely evaluated using the RoboFlag testbed. The research chosen for this paper provides an example of the depth and variety of applied theory that can be studied with this test system. While there are many examples that could be detailed which demonstrate the RoboFlag testbed, task allocation and operator decision modeling are two examples which would be difficult to mimic in other university based testbeds, both because of the numbers and realism of the robotics and the human interaction.

The discussion on the incorporation of RoboFlag in the task allocation studies is left for Chapter 2. However Section 1.4, demonstrates RoboFlag as a tool for decision modeling research focusing on the management of semi-autonomous agents. Particular attention is given to the relationship between the number of agents a human operator is responsible for managing and the degree that human operators trust and/or rely upon automation. A discussion on the nature of tasks that are turned over to automation as compared to the operator's desire to maintain an active role is also provided and supported with the results of sets of tests that use both single and multiple human operators. As decision modeling is often considered an interdisciplinary field that combines more traditional engineering with elements of cognitive engineering and computer science, this second study provides a contrast to the autonomous task allocation study to demonstrate the flexibility of the RoboFlag test system.

In addition to the discussion of the RoboFlag test system and studies presented here, the reader is encouraged to refer to http://roboflag.mae.cornell.edu for a more in-depth description of its most current features and operations guidelines as well as the most recent studies associated with the Cornell RoboFlag project.

1.2 RoboFlag Overview

RoboFlag was born out of the Cornell World Champion RoboCup teams led by Prof. Raffaello D'Andrea [1.17], from a desire to create a highly dynamic environment for a variety of multi-agent research. The test system can be described in terms of several different research projects that range from complex cooperative control studies [1.13],[1.15],[1.18] to operator tests and decision modeling [1.16],[1.19],[1.20]. As depicted in Figure 1.1, the standard RoboFlag scenario is the "Capture-the-Flag" game

consisting of two teams of robots that are in a competition to enter each other's territory, capture their opponent's flag, and attempt to bring the flag to their "home zone", all while defending their own flag and avoiding team specific "defense zones". [1.10]

Figure 1.1: left) Arbiter GUI of a RoboFlag Scenario, right) Blue Team HITL GUI of the Same RoboFlag Scenario. Standard RoboFlag Components are Labeled in Both left) and right). Note: The Blue Team HITL Only Displays what the Blue Team Agents Detect in their Sensor Cones

Each robot in the scenario is an autonomous decentralized agent running its own separate program. Coordination of multiple-agent activities and knowledge of its environment outside of its own sensors can only be obtained through communication with its teammates. All robots of a given team also send their sensor information to an optional human operator(s), referred to as the Human in the Loop (HITL), for which a GUI is shown in Figure 1.1. The HITL program offers options for centralized and/or semi-autonomous control of the robot agents. During the scenario, the HITL has the option to take control of any robot or group of robots on their team and change the artificial intelligence strategy that the robots are executing.

Although the dynamic and competitive nature of the standard scenario demonstrates some of the potential for studying multi-agent systems, the strength of the RoboFlag test system is in the generic nature and flexibility of its features. Each object in the RoboFlag system was designed to be representative of the common components found within many multi-agent areas of research. This intended relationship is outlined in Table 1.1. For example, the flags themselves merely function as a representation of an objective target which can be specific to a team or teams, to certain robot(s), and can be made to regenerate or change certain parameters with time.

Table 1.1: Representative meaning of RoboFlag common components

Robots	Agents
Flags	Objective Targets
Defense zones	Areas specific agents are not permitted
Home zones	Safe areas / Supply caches
Obstacles	Neutral Agents / Random Interference
Field	The boundaries of the scenario
Territories	Field sections specific to a set of agents

Each instance of these objects can also be made to have its own unique set of parameters. This flexibility can especially be seen in the heterogeneous nature of the robot agents themselves. In the RoboFlag v.3.0 system, each individual agent can vary according to a list of over 30 parameters, P_S. A complete list of these parameters is available at [1.16] and an abbreviated list is provided in Table 1.2 but they are responsible for specifying such features as the agents' sensors' range and noise, the

bandwidth and latency of the communication channels, and allows the agents to exhibit "rover-like" or "aircraft-like" dynamics.

Table 1.2: Abbreviated list of Parameters for RoboFlag Objects

Parameter Name	**Parameter Description**
V_{max}	object's maximum velocity
V_{min}	object's minimum velocity
D_{tag}	distance to another object where it can potentially tag another object
V_{tagMax}	maximum velocity this object can travel and be able to tag another object
$D_{visAngle}$	Angle from the object's zero degree rotation that the object's sensor cone extends
$D_{visRadius}$	Radius the object's sensor cone extends from the object's center
D_{refuel}	Maximum amount of fuel this object can carry
V_{rotMax}	Object's Maximum rotation velocity
Shape	Convex polygon shape the object has
BasicType	Common component of this object (agent, flag, defense zone, etc.)
Tracking	Constant for reducing the position, p, uncertainty of an observed object
Identify	Constant for reducing the type, Ty, uncertainty of an observed object
TrackingMult	Constant for reducing the rate that position, p, uncertainty is reduced when this object is observed
IdentifyMult	Constant for reducing the rate that type, Ty, uncertainty is reduced when this object is observed
Transparent	Object is not displayed in GUI
AlwaysVisible	Object is always displayed in GUI and reported in agent's local sensor data, s_L, even when out of the agent's sensor cone's range
TypeColor	Color displayed when drawing this type
T_{grace}	Time for a rule to hold true before that rule is applied
$D_{tagAngle}$	Max angle from this object's zero degree rotation that the object can tag another object
TagProbability	Probability that if the rest of the tag rule is true that a tag will be applied
"StartingPosition"	Starting position of the object, specified from the object's center
InitialUncertainty	Maxmium position and type uncertainty that this object can be observed as having
LocationFunction	Method by which this object's position is reported i.e. by its center ,by upper left coordinate, etc.

Furthermore, the most recent release of RoboFlag extends this flexibility to allow every object to have a unique state machine, commonly referred to as the object's rules, that defines how this specific object can influence other objects within the RoboFlag environment. More on the state machines is provide in Section 1.3 , but as is depicted in Figure 1.2, this toolbox of environmental components allows for a very wide scope of research to be explored.

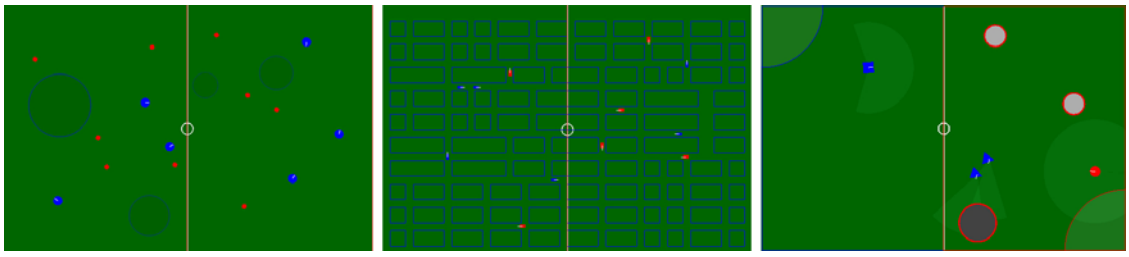

Figure 1.2: RoboFlag Screenshots from: left) an Optimal Task Allocation Study, center) an Initial DARPA Grand Challenge Study, right) an AFRL Decision Modeling Study with High Levels of Uncertainty

1.3 RoboFlag Architecture

With the aid of Figure 1.3, this section details the specific role of the six basic components of the current RoboFlag test system. Particular attention is given to the flow of information between the components which is set to run at a rate of 30 frames per second, where a frame is one complete cycle in the flow of information. The six primary components are:

A. Arbiter: the supervising program and global sensing information system

B. Network: the inter-program communication system

C. Agents: the individual robotic entity programs

D. Human-In-The-Loop (HITL): the optional human operator interface and/or centralized control program

E. Simulator or Robotics Hardware System: simulates the physics of the real robot system or carries out and observes the execution of the Agent software commands to the real robot hardware.

F. Logger: program the produces records of the scenario's events

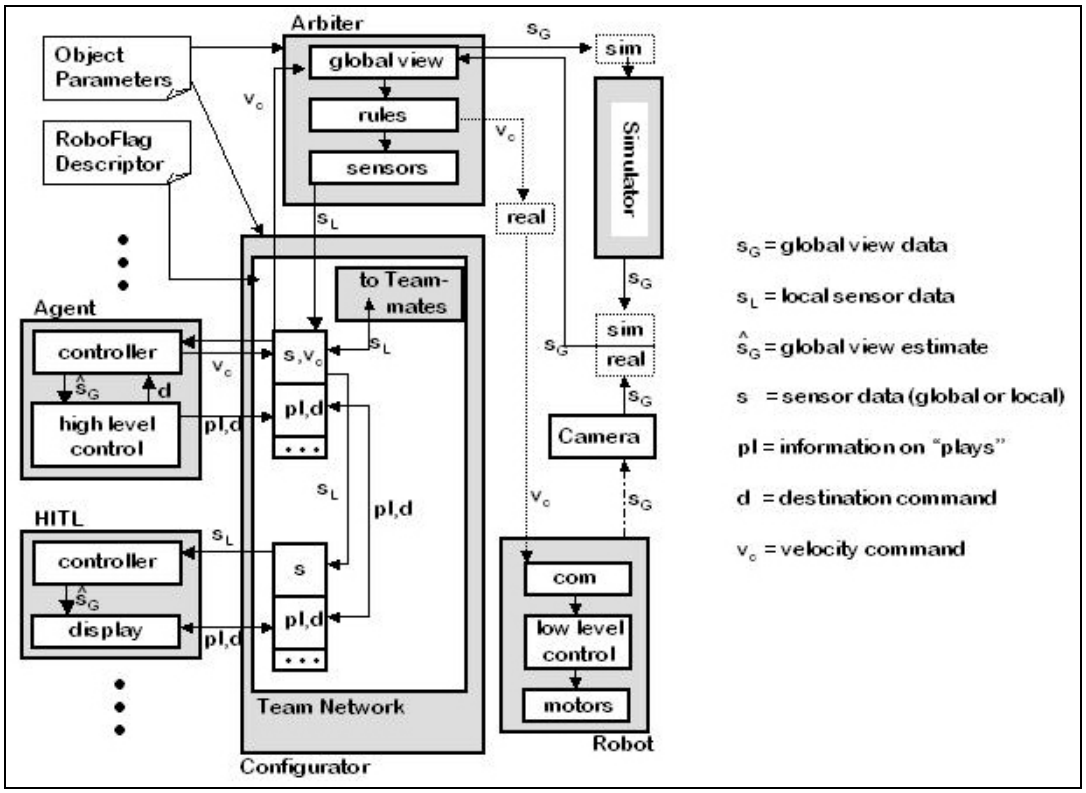

Figure 1.3: RoboFlag v3.0 System Architecture

1.3.1 Arbiter

The Arbiter program oversees the entire RoboFlag test system. Its primary responsibilities include the initialization of the scenario, maintaining a record of the current condition of all objects in the environment, i.e. the global view, s_G, enforcing

the objects' scenario state machines as the defined set of "rules", *Ru*, and if required, creating a filtered version of the global view to match each agent's simulated sensors, i.e. the agent's local sensors, s_L. The Arbiter also handles the communication with the Robotics Hardware system which is discussed later in Section 1.3.5.

1.3.1.1 Initialization:

The initialization of a RoboFlag scenario is handled through the instructions specified in a single text file named *ObjectParameters*. In this text file, all of the objects that make up the scenario, S_{obj}, are defined in terms of a type, *Ty*, and as belonging to certain team, *Tm*. The type, *Ty*, definitions specify the common component (Robot, Flag, Field, etc.) as well as set the variable parameters, P_S, introduced in the previous section, and the specific scenario state machine or rules for every type, Ru_{Ty}. These definitions also include a unique identification number, *ID#*, and name, such as "Landrover", "Jet" or simply "Circle", for the purpose of later indicating to the operator the intended object representation. Multiple objects within the scenario can then be declared to be of any one of the defined types, in a similar fashion to declaring instances of a class. More general scenario parameters, such as object starting positions and the length of the time permitted for the scenario, are also specified in this file along with communications settings that will be mentioned in the Network section.

1.3.1.2 Global View:

The global view, s_G, of the scenario is a frame specific snapshot of the current condition of all objects within the RoboFlag environment, S_{obj}. This snapshot includes the information expressed in equation (1.1) for every object, where *ID#* is a unique identification number for every member of a team, *Tm*, the variable *pos* is the position,

o is the orientation, vel is the velocity, $\underline{ss_t}$ is the scenario state, and f is the current fuel level which is a parameter that can be defined for every type, Ty. The rest of the type parameters, P_S, do not need to be included in the snapshot as they are encapsulated in the value of Ty . Fuel, f, remains in s_G partially for backwards compatibility. It should be noted that s_G does not contain information regarding the higher level control / agent artificial intelligence as this is maintained within the Agent programs themselves.

$$s_G = \left\{ \left\langle Ty, Tm, ID, pos, o, vel, ss_t, f \right\rangle_i \right\} \quad \forall i \in S_{obj} \tag{1.1}$$

1.3.1.3 Scenario State Machine / Rules:

Before sending sensor information onto the Agents, the Arbiter uses the global view, s_G, to examine the scenario state machine for all objects, Ru_{Ty}, and determines if a state change is required. The scenario state and rule system is described in detail for clarity and to demonstrate the flexibility that makes RoboFlag a unique test system. The RoboFlag test system uses a basic four state system for all objects. The four available states are: `active`, `flagged`, `tagged`, and `inactive`. The RoboFlag system allows the researcher to have some control over the interpretation of these states but the default meanings are listed below with key points highlighted in Table 1.3:

- `Active` is the default or normal operation state. `Active` objects running an Agent program receive local sensor information s_L, and can communicate with other agents and their team's HITL(s).
- Objects that have reached an objective target, i.e. "flag", become `flagged` and remain `flagged` until another state change. By default, `flagged` "flags" also modify their position to match the position of the object that

caused the flag to change to a `flagged` state. `Flagged` objects running an Agent program also receive local sensor information s_L, and can communicate with other agents and their team's HITL(s).

- `Tagged` objects are temporarily non-active and placed under the direct control of the Arbiter until another state change occurs. The default Arbiter control is to return the `tagged` object to its initial position and state. `Tagged` objects do not receive s_L and cannot communicate with other agents or the HITL(s).
- `Inactive` objects are more permanently non-active and are placed under the direct Arbiter control to maintain their current position, *pos*, at the time of the state change. The option to change the state of an `inactive` object is easily available but the default operation is that `inactive` objects remain `inactive` for the duration of the scenario. `Inactive` objects do not receive local sensor data, s_L, and cannot communicate with other agents or the HITL(s).

Table 1.3: Summary of key default significance of the four available scenario states, * if object is running an Agent program

Scenario state (ss_t)	Follow Agent Program*	Communicate with other Agents and the HITL*	Receive local sensor information*, s_L	Under Arbiter Control	Maintain position
`Active`	X	X	X		
`Flagged`	X	X	X		
`Tagged`				X	
`Inactive`				X	X

As mentioned above, the decision to change an object's state is based upon the state machines' rules, Ru_{Ty}, specified in the *ObjectParameters.txt*. Each rule must include the type of the object that causes the rule change, Ty_A, and the type of the object whose state changes if the rule is applied, Ty_B. With these types specified the rule can then be defined by a state change number, r_{ss}, a weight, r_w and a list of functions, r_f. The state change number, r_{ss}, is used to indicate what state change will occur to the Ty_B object if the rule is applied. Furthermore, each state change number is specific to allow only changes from a designated initial state, $ss_{t,i}$ to a final state, $ss_{t,f}$. More than one rule can have the same state change number to allow there to be more than one way to cause the same state change. For example, it may be desirable to have two rules for an active agent to become inactive.

As part of determining whether a rule is applied, the arbiter stores a separate variable $r_{sum,ss}$ for every rule number r_{ss} within an individual object's type's rules, Ru_{Ty}, for every individual object, *obj*, that is a part of the scenario, S_{obj}. At the beginning of each frame, all individual $r_{sum,ss}$ numbers for all objects are reset to zero. Then to determine if a rule is applied, all functions within the rule's list, r_f, must return true for an individual object. Then and only then is the rule's weight r_w added to the individual object's $r_{sum,ss}$ as is expressed in equation (1.2). Once all rules have been examined for all objects, if the value of $r_{sum,ss}$ for any individual object is greater than or equal to one, the states change of the associated r_{ss} is applied as shown in (1.3).

$$obj\left(r_{sum,ss}\right) = \left(\sum_{r_{ss}, \forall obj \in S_{obj}} r_w r_f\left(obj\right) \right) \tag{1.2}$$

$$obj\left(r_{sum,ss}\right) \geq 1 \;\; \Rightarrow \;\; obj\left(ss_{t,i}\right) \xrightarrow{r_{ss}} obj\left(ss_{t,f}\right) \tag{1.3}$$

For example, if an `active` agent of Ty_B should become `tagged` when it is contacted by an opponent of Ty_A but only if the agent is in the opponent's territory of Ty_C, two rules must be set up. The first rule is specified for the type of the opponent agent, Ty_A, and uses two functions to check if the specific Ty_B agent is 1) `active` and 2) in contact with the opponent. The second rule is specified for the type of the opponent's territory, Ty_C, and uses two functions to check to see if the Ty_B agent is 1) `active` and 2) within the opponent's territory. Each function, however, has a rule weight r_w of only 0.5. This means that both lists of functions must be true with regards to the same type Ty_B object in order for the rule to be applied and for that Ty_B object to be changed from `active` to `tagged`. A complete list of the currently available state change functions are available at [1.16].

1.3.1.4 Local Sensors:

Every object within a RoboFlag scenario that is running a separate Agent program is capable of receiving sensor information from the Arbiter. The default sensor information sent to an Agent is expressed in equation (1.4), which is very similar to the global view in equation (1.1) with the exception that data is only given for objects that are considered to be within the Agent running object's local sensor range, L. The standard way of setting an object's sensor range is through two of the object's type parameters, DvisRange and DvisAngle. Together these two parameters establish a sensor cone with a radius of DvisRange from the object's center and DvisAngle to either side of forward orientation of the agent. Any object located within the sensor cone, as determined by the Arbiter, can be sent to the object's Agent program as a member of L.

$$s_L = \left\{ \langle Ty, Tm, ID, pos, o, vel, ss_t, f \rangle_i \right\} \quad \forall i \in L \tag{1.4}$$

Whether this data is sent to the Agent program is determined according to the object's state as listed in Table 1.3. Before the local senor data, s_L, is sent, the arbiter may also "dirty" the data, adding noise and/or uncertainty to the position, *pos*, and even type, *Ty*, of the reported object. If RoboFlag is run in simulation, noise is added by default to s_L to match the noise seen in the Real Robotics Hardware. More on the use of the local sensor data, s_L, is provided in Section 1.3.3, *Controller*, and discussion of the uncertainty options is provided in Section 1.4.

1.3.2 Network

The Network is the communications hub for the RoboFlag test system and is actually a sub-system of several programs consisting of a Configurator program and a separate Team Network program for every team. The Configurator program is the first program to be run in starting the RoboFlag test system and creates the communication handles in between all of the programs. The Configurator program begins by reading the *ObjectParameters* text file which, in addition to the scenario initialization information mentioned in the Arbiter section, lists all of the IP addresses of the computers that run the Arbiter, Team Network, and Simulator programs. The *ObjectParameters* text file also lists the network ports that will be used for communication with each program.

The Team Networks programs are started next and each of these act as a separate hub for communications between the Agent programs and between the Agent and HITL programs of a given team, along with communication between Agents and the Arbiter or Simulator. The connection to the Team Network from the Agent and

HITL programs is done through a separate thread in these programs called the Agent Interface. Communication between the Agents and the Agents and HITL is limited though, typically through simulation, according to three governing parameters.

- $Bw_{i,j}$: the maximum available bandwidth available for communication between Agent/HITL *i* and Agent/HITL *j*.
- $La_{i,j}$: the communication latency between Agent/HITL *i* and Agent/HITL *j* in frames.
- Ch_k : for every available $Bw_{i,j}$, the bandwidth is split into up to *k* message channels, where Ch_k can vary from 3 to the number of bytes in $Bw_{i,j}$. Every channel uses 2 bytes for message formatting / time stamping and the rest of the channel is available for the researcher to specify.

Values for $Bw_{i,j}$ and $La_{i,j}$ for all Agents and HITL programs are listed in the *RoboFlagDescriptor* text file and are read in by the Team Network programs upon startup. The division of the available $Bw_{i,j}$ into the message channels, however, is handled within each Agent or HITL program. By default, the first channel, referred to as Ch_0, is set up as a priority channel to send/receive sensor and velocity command information. The size of this priority channel must also be set to be at least the maximum size of the sensor and velocity command information that can be sent to ensure that this information can be exchanged at a rate of 30 Hz.

All other message channels, however, can be set to be a smaller size than the desired communicated data. In these cases, where the desired communicated data is greater than the size of the k^{th} message channel, Ch_k, the desired data is divided into Ch_k sized segments. Then every frame, one segment is sent until the entire message

has been sent. Assuming that no previous messages are being sent, a message of size *Ms*, is then communicated in, F , frames, according to equation (1.5).

$$F = Ms \big/ Ch_k + La_{i,j} \tag{1.5}$$

To prevent large messages through small channels causing a significant backlog, there is an option to delete the first message or overwrite the last message if the message queue becomes full. However, as both $Bw_{i,j}$ and Ch_k are specified by the researcher this situation can be easily avoided. For messages that are commonly split amongst several frames, an optional messaging confirmation system is also available for the non-priority message channels. This system continues to try to send a message until the receiving program returns a confirmation of delivery. However, if another message is waiting in the queue, the confirmation message system adds one frame of latency to sending the next message.

The communication to the optional Robot Hardware system, however, is not handled through the Network but is handled through the Arbiter as will be described in Section 1.3.5.

1.3.3 Agent

The Agent program is comprised of three threads, an Agent Interface thread, a Controller thread, and a High Level Control thread. The Agent Interface thread is used to connect with the Team Network programs and handles the sending and receiving of messages. Details on the message system can be found in the Network Section 1.3.2. The Controller thread is responsible for interpreting and building the priority channel

messages that are sent/received by the Agent Interface. The Controller thread also handles sensor fusion methods to build an estimate of the global view, $\hat{s}_G$, and the conversion of High Level Control destination commands, *d*, into velocity command, vel_c, that can be sent to the Simulator or actual Robotic Hardware via messages to the Arbiter. The High Level Control thread is responsible for determining the actions of the agent given the estimate of the global view, $\hat{s}_G$, as well as interpreting and building non-priority channel messages to and from its teammates' Agent and HITL programs. All of the code for the Agent and HITL programs, however, is made available to the researcher to change to meet their needs.

1.3.3.1 Controller:

The Controller thread is a linear loop that, if its scenario state allows, begins by receiving its local sensor data, s_L, from the Arbiter as previously expressed in equation (1.4). Once received, the Controller also sends its s_L to its teammates and receives analogous information from its teammates. However, due to the researcher set latency of the Network, $La_{i,j}$, the timestamp of various received s_L data will most likely not coincide. This leads to the need of sensor fusion methods for every agent to build a global view estimate, $\hat{s}_G$. As the team, *Tm*, and *ID#* of every object is provided in the local sensor data, s_L, sensor fusion can be easily performed by simply matching these parameters and taking the data with the most current time stamp. The researcher is free to define a replacement for this sensor fusion approach and the *ID#,* team, *Tm*, *Ty*, data that is passed along is quite often ignored in sensor fusion studies. For example, another geometric based sensor fusion algorithm is also available and is detailed by [1.11],[1.16]. For further complexity, the current RoboFlag system also offers the option to provide uncertainty bounds on the position and type of an object which is discussed further in Section 1.4.

1.3.3.2 High Level Control:

In addition to receiving the global view estimate, $\hat{s}_G$, from its associated Controller, the High Level Control listens to receive commands from the HITL and its teammates via the non-priority channels. Using these inputs, the High Level Control runs a researcher defined, study specific set of artificial intelligence or strategy code whose purpose is to output a desired destination, *d*, for the agent. Included in the most current release of RoboFlag, is an artificial intelligence system designed for the standard scenario summarized in Section 1.2. and detailed in [1.10]. This system is commonly referred to as the scenario "Playbook" and each separate strategy within the system is a "play". Information concerning the setup and/or execution of the plays, *pl*, is also passed onto its teammates and the HITL via the non-priority messaging channels. The playbook system and the communication of play data, *pl*, are designed to be easily modified for a number of different research studies as is shown in the examples of Section 1.4.

1.3.4 HITL

The HITL program is comprised of three threads, an Agent Interface thread, a Controller thread, and a Display thread. The HITL Agent Interface thread and the Controller thread are very similar to the Agent's Agent Interface and Controller threads. The HITL Controller however does not receive sensor data directly from the Arbiter but receives only the local sensor data, s_L, from its teammate Agent programs. In addition, as with the Agent program's Controller, the HITL Controller uses the same sensor fusion method to determine a global view estimate, $\hat{s}_G$.

The Display thread draws the global view estimate, $\hat{s}_G$, on the GUI along with data such as the fuel or scenario state of the objects contained within $\hat{s}_G$. In addition the Display thread may draw information regarding the plays being run by different Agent programs and those agent's destinations, *d*, which are received via non-priority message channels. The Display thread also is responsible for listening to user inputs, such as the keyboard and mouse, and operates a number of GUI buttons, checkboxes and menus. These input devices in turn can be used to change the elements displayed by the GUI and/or send messages to the Agent programs to cause a change in the plays or the performance of those plays.

1.3.5 Simulator / Robotic Hardware System

The RoboFlag system is designed to execute the velocity commands, vel_c, that are the output of the Arbiter via either the Simulator program or the Robotic Hardware System. The Arbiter receives the velocity commands, vel_c, from Agent programs and generates velocity commands for Arbiter controlled objects, such as `tagged` agents for example. If the scenario is being run in simulation, the Simulator program receives vel_c and the global view estimate, $\hat{s}_G$, to modify the position of the agents according to equations (1.6-1.8) and equation constraints (1.9,1.10) where the U_x, U_y *and* U_θ terms are internal control inputs that would be used in the robots' low level code [1.9],[1.10]. It is noted that the Simulator uses the same equations as the low level controller code to ensure that the tracking of input velocity commands are as similar as possible to the real Robotics Hardware System. A delay time of T_{delayOut} is also added to the execution of the commands to simulate the communications delay between the arbiter and the real Robotics Hardware System. Formally the dynamics are written as:

$$\ddot{x}(t) = \alpha U_x(t) - \beta \dot{x}(t) \tag{1.6}$$

$$\ddot{y}(t) = \alpha U_y(t) - \beta \dot{y}(t) \tag{1.7}$$

$$\ddot{\theta}(t) = \alpha U_\theta(t) - \beta \dot{\theta}(t) \tag{1.8}$$

$$U_x + U_y \leq 1 \tag{1.9}$$

$$|U_\theta| \leq 1 \tag{1.10}$$

If the scenario is connected to the Robotics Hardware System, the velocity commands, vel_c, are sent from the Arbiter via a wireless transmitter to the communication or "Com" system receiver on the robot's themselves. These commands are then given to the robots' low level controller which is based on equations (1.6-1.10). This in turn generates control inputs for the actual motors. Details on the Cornell robotic designs can be found at [1.16],[1.20]. As a real hardware system is being used to carry out the commands, the global view, s_G, of the Robotics Hardware System is determined using a system of overhead cameras. This vision system is also detailed in [1.16],[1.20], but the output global view, s_G, is sent to the Arbiter computer at a rate of 30Hz.

The RoboFlag Simulator has been designed to be a virtual replica of the experimental hardware, thus allowing a wide ranging set of experimental simulations to occur, both at Cornell and other institutions as is shown in Figure 1.4. A comparison of the simulator and hardware is detailed in [1.15].

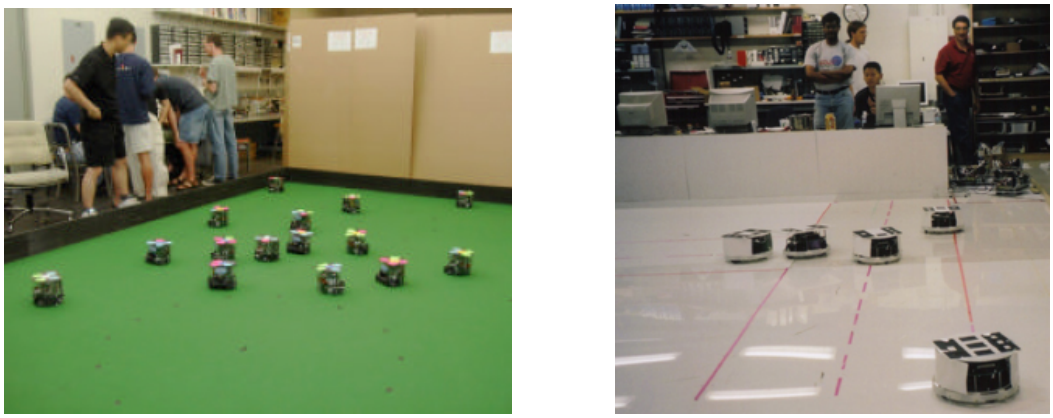

Figure 1.4: RoboFlag Implemented on

left) The Cornell Robotics Hardware, right) The Cal-Tech MooreBots

1.3.6 Logger

The Logger is an optional program within the RoboFlag system and is used to create XML files that record events or summaries of the frames from within the Arbiter and/or HITL programs. Communication to the Logger is also set up by the Configurator. Wrapper functions are provided within RoboFlag v.3.0 to aid the researcher in formatting the output files and a sample parser is provided to aid the researcher in making use of these files. Details on the use of the Logger can be found at [1.16].

Researchers using the RoboFlag test system are encouraged to refer to http://roboflag.mae.cornell.edu for more in-depth descriptions of its most current features and operations guidelines.

1.4 Cognitive engineering Experimental Studies

The RoboFlag test system has been used for a very wide variety of cognitive engineering studies [1.6],[1.15],[1.19]. This section illustrates the use of RoboFlag for

studies of this nature and presents a discussion on one set of studies which focuses on trends in human controlled automation of multiple robots. From the overview it is readily apparent that the optional Human-In-The-Loop (HITL) operator GUI of the RoboFlag test system could be used to study the understanding of the effects of situational awareness and GUI design as many cognitive engineering studies tend to focus on [1.15],[1.19],[1.21]. For this reason, this section discusses components of cognitive engineering research that better demonstrate the flexibility of the RoboFlag system and take advantage of the reproducibility of the scenario set-ups in a cognitive engineering research setting. The study presented in this section examines how changing the number of agents an operator is responsible for managing influences the reliance, trust and/or use of automation as well as the operator's ability to meet the scenario's objectives. The discussion of this study then motivates a second study where the scenario is complicated by the presence of greater uncertainty in the information presented to the human operator.

The first study used the standard RoboFlag scenario in creating the operator's objective, as is explained in the RoboFlag Overview of Section 1.2 and in [1.10]. This scenario is essentially a timed game of "Capture the Flag" where the role of the HITL operator is to evaluate the environment via the HITL GUI and manage their team of agents to capture the opponent's flag and protect their own. Additional attributes include avoiding being tagged by the opponents, and refueling when fuel was low. The operators control of the agents was in the form of switch the artificial intelligence strategy of the agents and/or taking direct manual "joystick" control of any agent as well.

Each different artificial intelligence strategy that was made available to the HITL was referred to as a 'play' and the set of all 'plays' was the 'playbook'. Some of the abbreviated names of the plays within the playbook that the operators controlled included those listed below. Details on other plays that were available are found in [1.15],[1.16].

- *Guard*: agent pursues "taggable" visible opponents within a certain radius of a given position
- *Circle Offense*: agent attempts to circle though the opponent's territory looking for opportunities to steal the opponent's flag and return it
- *Patrol*: agent moves back and forth along the inside of their territory's boarder
- *Low Fuel Go Home*: an override setting that activated when the agent's fuel was within 20% or less of the fuel required to reach their home zone to refuel. Upon activation, the agent stops their current play and returns home

In each set of games, the number of agents per team, n_a ,was varied to either 2,4, or 6, and the maximum velocity of all agents in the scenario, vel_a, was either 0.25 m/s, 0.5 m/s and 0.75 m/s as is expressed in equation (1.11). A test set, $\tau_{i,j}$, is defined as a collection of games where each game has i agents per team and each agent has a maximum velocity of j. Each operator participant was asked to participate in a set of every possible combination of number of agents, n_a, and maximum velocities, vel_a, as varying n_a and vel_a in turn varied the amount and rate of information the operators were responsible for interpreting and responding to.

$$n_a = \{2,4,6\} \quad , \quad vel_a = \{0.25,0.5,0.75\} \tag{1.11}$$

Using the *ObjectParameters* text file, the same scenario could be created and run repeatedly for multiple operators with the assurance that only the n_a and vel_a values could be modified for different test sets without affecting the consistency of the rest of the scenario set-up. Furthermore, using the RoboFlag system allowed the studies to standardize such elements as the reaction methods of autonomous units, i.e. the plays. Other parameters that are commonly varied in RoboFlag studies, such as changing the scoring system to shift operator priorities or setting the fuel of agents to different levels to require more of the operator's time to be focused on monitoring the condition of the agents or agent "maintenance", were kept constant in this study as well.

For each test set, $\tau_{i,j}$, 12 individual ten-minute games, with one human operator per team, were run in groups of four with a 15 minute break after each 4 game group. Two test sets were run per testing day for a total of six days of data collection. During the games, the RoboFlag Logger feature was used heavily to record a variety of elements, including the scoring as a measure of performance, the operator's mouse clicks as one indication of the workload, and the percentage of time each play automation was used versus the percentage of time the agents were under direct manual control. More traditional instruments such as the NASA TLX and SART survey tests were also administered to the operators after every game. A more complete list of the elements tracked via the logger and details on these survey tests can be found at [1.15].

A smaller second scenario of three test sets was also run where vel_a was fixed to the middle value of 0.5 m/s and n_a was varied. The difference in this scenario, however, was that each team had two human operators where each operator was not

only able to control any of the team's agents but the human operators were also able to communicate with one another. Some of the results of these tests are also discussed in the next section.

1.4.1 Experimental Results

Numerous trends were identified in the analysis of the log files from the 6 days of testing. One general trend was that there was an overall increase in the performance measure of the scores as the maximum velocity or the number of agents was increased. This was not surprising, however, as both of these resulted in more agent interactions in the same time frame and hence more scoring could occur. A more interesting result, however, is the effect of how changing the number of agents an operator is responsible for influences the reliance and trust of automation. These effects were determined though data evaluation of the play usage by each user, as shown in Figure 1.5.

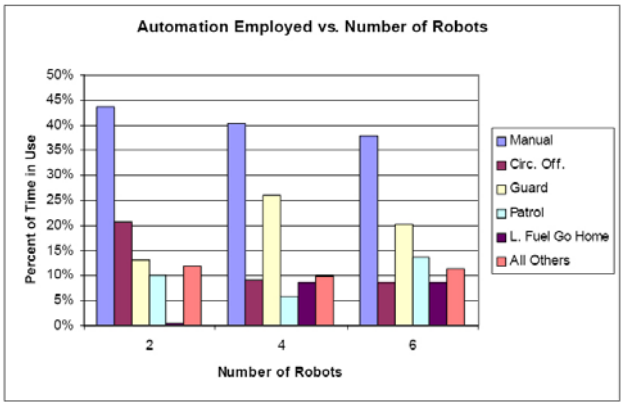

Figure 1.5: The Percentage Time in use of Automations or Manual Control as the Number of Agents, n_a, is Varied, with One Operator Per Team

This figure summarizes all of the single operator test sets with respect to the amount of operator time that was dedicated to manual control, $\%t_{MC}$, as compared to

the amount of time the agents were given commands via the automation plays, $\%t_{AC}$. In all cases, automation commands were used more than 50% of time and in general as the number of agents, n_a, was increased, the percentage of time the agents were running automation commands also increased. These results intuitively point to the users requiring more heavily on automations in order to achieve the end goals as the workload increases, as defined by the number of robots being controlled.

Another element captured in Figure 1.4 is the switch in operator preference of the automations as the number of agents, n_a, increased. One readily apparent switch is the increase in the use of the low fuel go home play. As mentioned in the previous sub-section, this play automatically recognizes when an agent should return home to ensure that the agent does not run out of fuel. Refueling the robot agents was an important task, as agents that ran out of fuel became inactive and were permanently out of the game. Although this did not directly influence the objectives, it was a significant maintenance task that required constant monitoring of the agents. Monitoring only two agents was relatively easy, but as n_a increased, operators found it easier and effective enough to hand this task off to the automation. This in turn allowed the operators to focus on the more direct objective of capturing the flag.

This concept of the effectiveness of the automations also had a significant impact on the use of automations. Many times operators would begin by giving automation commands to all of the agents. However, as the agents approached an objective such as tagging an invading opponent or capturing the flag, the operator would switch to manual command, regardless of whether the automations would be successful on there own. Part of the reason for this behavior was due to a definite

learning curve for the operators in being able to trust and understand the capability of the automations, as was brought out in the TLX and SART surveys.

The more significant finding from these surveys, that is also supported by Figure 1.5, is that there was a distinctive desire of the operators in all test sets to maximize their participation. In any situation, the operators continually wanted to be the driving force behind the success of their team. This can be seen in Figure 1.5, where the majority of the automation used is for the more reactive / monitoring task of defensive and the operators preferred to take command of the offensive tactics where there is the greatest potential to score the most points, even though the offensive automations were regularly successful in their objectives. This desire to be in constant direct control instead of managing the overall action was further supported in the test sets of two operators per team as is shown in Figure 1.6.

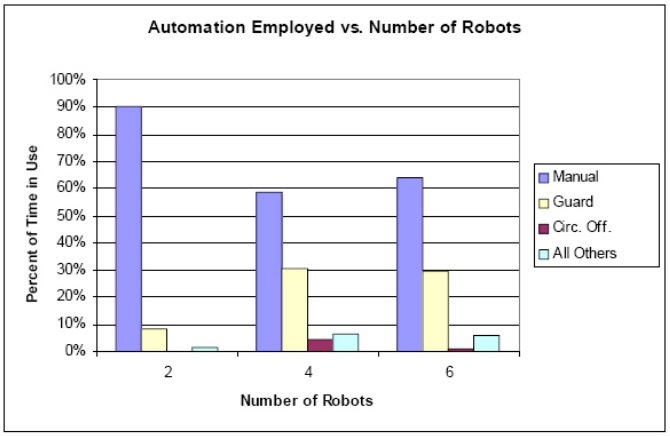

Figure 1.6: The Percentage Time in use of Automations or Manual Control as the Number of Agents, n_a, is Varied, with Two Operators Per Team

From a performance standpoint, the scores in the two operator test sets were slightly higher than in the one operator tests and the data for these scores can be found in [1.15]. This difference can be attributed to the ability of the operators to cooperate. Furthermore, as the operators learned to cooperate, as was the case in the final set of games of $n_a = 6$, the amount of manual control actually increases over the $n_a = 4$ set. This supports the intuition that the operators have learned how to operate together better and have found ways to increase their active participation without sacrificing performance.

Overall in this study, a general trend was found that an increase in the number of agents, n_a, causes operators to turn over some their responsibilities to automations, while still attempting to maximize their own participation. The tasks that operators typically use automations for are the maintenance (refueling) and reactive monitoring (defensive) tasks, saving the tasks that are more proactive and oriented toward the direct objectives for manual control.

This study now provides motivation for a new scenario which more directly targets the connection between use of automation and performance by examining a scenario which forces the operators to have only manual controls commands, only automation commands, or an adaptive mix of the two. This new scenario, which is of particular interest to Dr. Scott Galster's group at Wright Patterson AFRL , focuses on a cooperative reconnaissance and identification mission, where the user controls multiple, heterogeneous vehicles for the end goal of searching and identifying all targets in an area. Uncertainty is also introduced into this scenario through the recent addition of RoboFlag's uncertainty type parameters which are set via the *ObjectParameters* file. Through these parameters every defined type, *Ty*, can specify

that uncertainty be added to objects reported in other objects' local sensors, s_L. This feature can then be used to report bounds on the object's observed position or even in estimations of the observed type, *Ty*, of the objects themselves. Furthermore, every type has parameters to define its sensors' ability at reducing uncertainty allowing for scenarios where faster "scout" agents with air-craft like dynamics are used to roughly identify the location and/or type of an object. Then slower "rover-like" agents will be moved in to more accurately pinpoint the information. As proof of this concept, an image from a beta-test scenario is shown in Figure 1.2c of Section 1.2. Dr. Scott Galster's group at Wright Patterson AFRL will be experimentally test the scenario using the AFRL subject pool. The Cornell team will receive all test data from AFRL, and then attempt to model all user decisions during the scenario with methods such as those described in [1.22].

1.5 Conclusions

The RoboFlag test system is a highly flexible research tool, well established in demonstrating the value and validity of multi-agent cooperative controls theory through experimental application. As testament to this ability, the theory behind the new optimal task allocation method, G^*_{TA}, was translated into a RoboFlag set of experiments. Out of these experiments, G^*_{TA} was shown to produce identical optimal solutions in computation times two orders of magnitude faster on average than a traditional MILP technique. Furthermore, the RoboFlag test system was also shown to be a valuable environment for Decision Modeling studies, particularly in focusing on the management of semi-autonomous multi-agent systems.

REFERENCES

[1] Chandler, P., Pachter, M., *et al.* "Distributed Control for Multiple UAVs with Strongly Coupled Tasks," AIAA *Guidance, Navigation, and Control Conference*, August 2003

[2] Bellingham, J., Tillerson, T., Richards, A., How, J., "Multi-Task Allocation and Path Planning For Cooperating UAVs" *Conference on Coordination, Control and Optimization*, Nov. 2001

[3] E. Fiorelli, N.E. Leeonard, P. Bhatta, *et al.* "Multi-AUV Control and Adaptive Sampling in Monterey Bay," *Proc. IEEE Autonomois Underwater Vehicles 2004: Workshop on Multiple AUV Operations*, June 2004

[4] Campbell, M., "Planning Algorithm for Multiple Satellite Clusters," *Journal of Guidance, Control and Dynamics*, Sept-Oct 2003.

[5] D. Dyke Weatherington, "DoD UAV Roadmap", *U.S. Department of Defense*, 2003.

[6] R. Parasuraman, S. Galster, and C. Miller, "Human Control of Multiple Robots in the RoboFlag Simulation Environment" *IEEE International Conference on Systems, Man, and Cybernetics*, Washington DC, Oct 2003.

[7] D. Schneider, A. Hoffman, C. Edmonds, *et al*. "Adaptive Sensor Fleet Development of Inexpensive Multi-Agent Robotic Basis System using the NASA Multi-Purpose Exoterration for Robotic Studies" *NASA Advanced Automations and Architectures,* May 2006

[8] R. D'Andrea, "RoboCup" National Science Foundation Directorate for Engineering Research Highlight Series, April 2001

[9] R. D'Andrea and M. Babish, "The RoboFlag Testbed", Proc. of the American Controls Conference, June, 2003

[10] R. D'Andrea and R. Murray, "The RoboFlag Competition", *Proc. Of the American Controls Conference*, June 2003, pp. 667-671

[11] M. Campbell, R. D'Andrea, D. Schneider, A Chandry, *et al*. "RoboFlag Games Using Systems Based Hierarchical Control", *Proc. Of the American Controls Conference*, June 2003, pp. 650-655

[12] J. Ousingsawat, and M. Campbell, "Multiple Vehicle Team Tasking for Cooperative Estimation" *American Control Conference*, June 2004

[13] J. Sullivan, S. Waydo and M. Campbell, “Using Stream Functions to Generate Complex Behavior” 2003 *Guidance, Navigation and Control Conference*.

[14] M. G. Earl and R. D'Andrea, “A study in cooperative control: The RoboFlag Drill,'” *Proceedings of the American Control Conference*, Anchorage, Alaska 2002

[15] J. Veverka, and M. Campbell, “Experimental Study of Information Load on Operators in Semi-Autonomous Systems,” *AIAA Guidance, Navigation and Control Conference*, Austin TX, Aug. 2003.

[16] D. Schneider, (2006, April) “The RoboFlag Website,” *Cornell University*, Available: http://roboflag.mae.cornell.edu/

[17] O. Purwin, R. D'Andrea: “Cornell Big Red 2003”, in: Polani D., Bonarini A., Browning B., Yoshida K. (Eds), *Robocup 2003: Robot Soccer World CupVII, Lecture Notes in Artificial Intelligence,* Springer, Berlin, 2003

[18] A. Chaudhry, R. D'Andrea, and M. Campbell, “RoboFlag - A Framework for Exploring Control, Planning, and Human Interface Issues Related to Coordinating Multiple Robots in a Realtime Dynamic Environment”, *ICAR*, 2003

[19] P.N. Squire, S.M. Galster, & R. Parasuraman, “The effects of levels of automation in the human control of multiple robots in the RoboFlag simulation environment.” *Proceedings of the Second Human Performance, Situation Awareness, and Automation Conference*, 2004

[20] J. Lee, (2006 , December) “Cornell RoboCup,” *Cornell University*, Available: http://robocup.mae.cornell.edu/

[21] S.M.Galster, and R.S.Bolia, “D ecision quality and mission effectiveness in a simulated command & control environment”, *Proc. of the Second Human Performance, Situation Awareness, and Automation Conference*, 2004

[22] J. Veverka, and M. Campbell, “Operator Decision Modeling for ISR Type Missions,” *IEEE Conference on Systems, Man, and Cybernetics*, Oct 2005.

CHAPTER 2

REAL TIME OPTIMAL TASK ALLOCATION IN HIGHLY DYNAMIC ENVIRONMENTS USING NON-MILP METHODS

Of the methods developed for Optimal Task Allocation, Mixed Integer Linear Programming (MILP) techniques are some of the most predominant. A new method, presented in this paper, is able to produce identical optimal solutions to the MILP techniques but in computation times orders of magnitude faster than MILP. This new method, referred to as G^*_{TA}, uses a minimum spanning forest algorithm to generate optimistic predictive costs in polynomial time in an A^* framework, and a polynomial time greedy approximation method to create upper bound estimates. A second new method which combines the G^*_{TA} and MILP methods, referred to as G^*MILP, is also presented for its scaling potential. This combined method uses G^*_{TA} to solve a series of sub-problems and the final optimal task allocation is handled through MILP. All of these methods are compared and validated though a large series of real time tests using the Cornell RoboFlag testbed, a multi-robot, highly dynamic test environment.

2.1 Motivation

As process designs have expanded from single automated systems to multiple automated systems that work together, there has risen a strong need to develop methodologies that optimally solve a variety of task allocation problems in real time. One area that has had particular need for this kind of solution has been in the study of multi-robot systems which have ranged from such applications as space exploration [2.1],[2.2], to coordinated UAV control [2.3],[2.4] to even robotic soccer [2.5]. The potential value of such a method has been formally recognized by the U.S. Department

of Defense [2.6]. Out of this need, Cornell has developed the RoboFlag testbed to be used to study a variety of semi-autonomous and autonomous cooperative control problems including task allocation in highly dynamic environments. Of the studies performed in this testbed, this paper offers two new optimal task allocation methods, G^*_{TA} and G^*MILP, that will be shown to offer both new alternative optimal approaches to the task allocation problem and significantly computationally faster results than traditional Mixed Integer Linear Programming (MILP) techniques.

Prior to this study, two of the primary approaches to the task allocation problem were optimal Mixed Integer Linear Programming (MILP) techniques [2.4],[2.7] and Heuristic Approximation methods [2.8]. MILP based methods have been gaining popularity to solve a wide variety of planning problems because they are provably optimal and many of the problems are easily translated into the MILP framework. Recent developments using MILP for task allocation work include work being done at MIT, Wright Patterson AFRL, Berkley and several other leading universities [2.2],[2.9]-[2.16]. Furthermore, due to its wide use, Dr. Sangiovanni-Vincentelli's group at Berkeley has even published a classification of MILP representations of the problem [2.14]. Although MILP methods have been shown to be very successful in operations where there are no strict time restrictions [2.4], the largest disadvantage to MILP methods is there computation time and memory usage. Hence significant difficulties can occur when they are applied to highly dynamic environments where solutions must be obtained quickly and problems must be completely resolved very frequently.

Due in part to these significant solution time requirements, the other primary approach for solving task allocation problems are Approximation heuristic methods

which sacrifice the guarantee of optimality in return for faster computation times. This area of research in particular has been growing very quickly, particularly for less complex versions of task allocation problems. Many notable recent methods make use of a variety of algorithms from other fields including negotiation and even financial models as well and have been implemented by groups at Carnegie Mellon, Stanford, USC and LAAS-CNRS [2.17]-[2.21]. A Stanford / USC summery of many of the other current approximation methods as well as a suggested taxonomy for describing the many problem variations can be found in [2.8]

Recent work has tried to overcome the optimal MILP's computation time issues and the approximation method's lack of accuracy through combining the two to create more reliable approximations. One example of this is the coordinated reconnaissance work of Ousingsawat & Campbell which combined MILP with clustering techniques to reduce the problem size [2.22]. Another example is the intercept path planning work done by Earl & D'Andrea which used MILP not to determine optimal assignment but whether there existed an assignment that met a certain goal [2.15]. Most recently, Rathinam & Sengupta from Berkley have modified Held-Karp's lower bounds TSP method to handle the multiple depot, multiple salesman task allocation problem to a 2-approximation LP-relaxation method that forces fixed ending tasks [2.23]. Some of the most notable work though has been done at MIT's Space Systems Laboratory under Dr. John How, including a study of MILP experimentations where MILP is used to solve higher level problems at a lower frequency (on the order of seconds) and model predictive control (MPC) is used in between MILP based updates [2.9],[2.16].

As will be shown, the work discussed in this paper offers new contributions to both non-MILP and combined MILP areas of task allocation research. Unlike many non-MILP approaches however, the G^*_{TA} method still provides guaranteed optimality, and both the new G^*_{TA} method and the new combined G^*_{TA} / MILP method, G^*MILP, produce guaranteed optimal solutions in average computation times order(s) of magnitude faster than the standard pure MILP method. This paper verifies these statements with experimental results in Section 2.6 where the G^*MILP also shows promise for improved scalability over a traditional MILP implementation.

This paper begins by formally defining the task allocation problem and relating it to past task allocation studies and the taxonomy in [2.8]. In Section 2.3, a summary of the RoboFlag testbed is provided with regards to the overall implementation of the new methods. Then in Sections 2.4.1 – 2.4.6 the paper's first contribution, the development of the G^*_{TA} method, is described in terms of its two main components; Sections 2.4.1 – 2.4.4 provides an overview of the A^* framework and the resulting guarantee of optimality and Section 2.4.5 – 2.4.6 describes how a greedy method can be added to provide upper bound cost estimates to reduce the A^* framework's search space.

A standard implementation of the pure MILP solution to the problem is discussed in Section 2.5.1. Then Section 2.5.2 provides an explanation of the paper's second contribution of how the G^*_{TA} can be combined with MILP to create the second new method, G^*MILP. In order to evaluate the G^*_{TA}, G^*MILP and traditional MILP methods, Section 2.6 presents the results of a series of tests run in real time on the RoboFlag testbed, which compares all of the methods and their components based upon computation times.

These results then motivate the paper's third contribution. In Section 2.7, the optimal G^*_{TA} method is also shown to be able to provide "any-time" solutions as well and therefore could also be implemented as an approximation method if desired. To demonstrate this additional approximation capability, a second set of G^*_{TA} tests are presented which were run under very strict time requirements as small as 25 milliseconds per trial problem. Under these time constraints, the G^*_{TA} method is shown to produce "any-time" solutions with an average percent error of less than 2%.

Finally, to demonstrate that the new G^*_{TA} method, like many traditional MILP implementations, can also handle a wide range of task allocation problem variations other than those specifically defined in Section 2.2, an overview of common task allocation variations and how the G^*_{TA} can be appropriately adapted is provided in Section 2.8.

2.2 Task Allocation Problem Definition

The term "task allocation" is used to describe a variety of problems where there is a given set of tasks (a.k.a. targets, goals, etc) that must be performed and there is a given set of agents (a.k.a. robots, sources, etc) that can perform the tasks. Furthermore, there is commonly a different task to agent assignment cost or utility for each task/agent pair as well as similarly a different assignment cost for every task/task pair for transitioning from one task to another. The goal is then to determine which tasks should be assigned to which agents and equally importantly in which order, so as assign all tasks while incurring the minimal cost and not violating any additional problem constraints.

This very general task allocation problem as addressed in this paper can be best described more formally by first defining the terminology. Let *s* stand for the number of agents (or sources) that can be assigned tasks and let *t* stand for the number of tasks (or targets) that must be assigned to the sources. Also let the method of transitioning a source from its initial state to a target state and likewise the method for transitioning from one target state to another, be referred to as a "path". Finally, let the ordered set of paths assigned to a source be referred to as a "trip", where Tr_i stands for the trip of source *i*. As a result, all trips always begin with a source and then contains an ordered list of the targets assigned to that source.

Table 2.1: Summary of Key Problem Definition Terms

s	# sources
t	# targets
path	Method of transitioning for a source from its initial state to a target state or from one target state to another
trip	Ordered set of paths to be determined for each source
Tr_i	the trip of source *i*

As will be specified in Section 2.3, for each path it is also assumed there is a way to calculate a cost for using that path as well as a way to measure any resource(s) consumed while transversing that path; where each source has limited resource availability.

With these terms in place, the task allocation problem can be defined as given *s* sources, *t* targets and a set of all valid paths, the method must create a set of trips for the sources such that each target is assigned to or "visited by" at least one source

without violating the sources' resource constraints. The most general version of this problem, which is the focus of this paper, is that a source can be assigned multiple targets, any target can be assigned to any source, and the order in which targets are assigned to a source does influence the cost. In addition, all costs are assumed to be non-negative and the cost associated from transitioning from state A to state B does not have to be the same as the cost associated from transitioning from state B to state A. The problem definition is further generalized to allow it to be more applicable to more real world scenarios by allowing the triangle inequality not to hold. Finally, the problem goal is to minimize the overall combined cost of all of the sources' trips as expressed as (2.1)

$$J = \sum_{i=1}^{s} (J_i(Tr_i)) \tag{2.1}$$

where J stands for the total cost and $J_i(Tr_i)$ is the cost of source i's trip. In this problem, it is assumed that a "one way trip" is the default problem variation where a source may end its trip at any of the targets or not be assigned any targets at all.

In terms of [2.8]'s taxonomy, this problem is essentially a more difficult version of the Single Task Robots – Single Robot Tasks – Time Extended Assignment or ST-SR-TA case, i.e. a robot can handle several ordered tasks but only one at a time (i.e. single task robots) and the tasks only require a single robot's attention at some point in the task allocation (i.e. single robot tasks). (A fairly well known example of a ST-SR-TA problem is the ALLIANCE Efficiency Problem (AEP) first stated by Parker [2.24].) The reason that the problem here is a more difficult ST-SR-TA is because of the interdependency between the costs, i.e. whether a robot visits target A or target B first influences the cost to travel next to target C. With this observation it

becomes apparent that this problem is an instance of the Multiple Depot, Multiple Traveling Salesman Problem and is hence NP-Hard as was shown by Korte & Vygen in 2000 [2.25]. As discussed in Section 2.1, both MILP and approximation methods [2.7]-[2.9],[2.26] have been applied to similar variations, and therefore this is a good problem to demonstrate the abilities of the new G^*_{TA} and G^*MILP methods. Also, as the new G^*_{TA} and G^*MILP methods were designed to be very general and flexible, Section 2.8 addresses some variations on the specific problem described here as well as general trends in the new methods' ability to solve these variations.

2.3 The RoboFlag Testbed

The analysis of the new G^*_{TA} and G^*MILP methods was carried out using the Cornell RoboFlag testbed. The RoboFlag testbed has already been used in a number of studies that have ranged from cooperative control [2.27]-[2.29] to operator tests and decision modeling [2.30]-[2.32]. The most general use however is to simulate two teams of up to 6 omni-directional robots each set against each other in a game of capture the flag [2.33]. Although the testbed is mainly used in simulation, numerous experiments have been run using RoboFlag to control real robots. In [2.31] the simulations were shown to be comparable to the physical robot game results. The testbed also allows for a range of field and robot parameters to be varied; for example, to allow "jeep-like" or "aircraft-like" robot dynamics. In the tests presented in this paper, all of the default parameters were chosen as specified in [2.33]. In order to understand how the task allocation algorithms are implemented in RoboFlag, a simplified overview of the main computational stages for each robot is provided below in Figure 2.1 and its supporting outline:

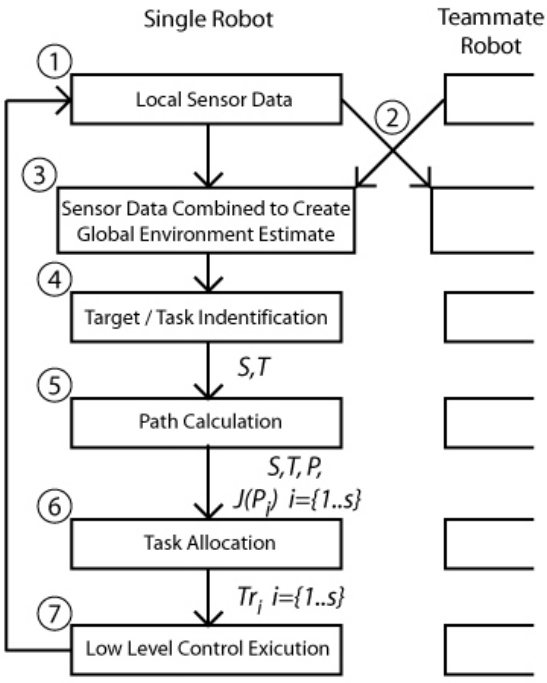

Figure 2.1: Computational Stages of the RoboFlag Testbed

1. Local sensor data is collected providing information on the robot's global position, orientation, and velocity, as well as the global position, orientation, and velocity of other objects in the robot's sensor area.[2.33],[2.34]
2. The local sensor data is sent out to teammate robots. [2.34]
3. The local sensor data of the robot's teammates is received and this data is combined with the robot's own local sensor data to develop an estimation of the environment, i.e. the RoboFlag field [2.35]
4. Strategy code determines a set of targets based upon the estimation of the environment. [2.34]
5. A Primitives based path planning algorithm [2.22] is used to calculate the lowest possible cost set of way points between any two targets (a "path") as well as between the robots' starting locations and the targets.

6. At the highest level planning, the costs of the paths are used to solve the task allocation problem optimally and an ordered series of paths ("trip") is generated for each robot.
7. Based upon the trips generated, way point destinations are sent to a lower level controller thread, which in turn converts the destination into velocity data to be used in the lowest level robot motor control at a rate of 30 Hz [2.34].

The methods described in this paper focus on stage 6. However, this modular nature is important as it not only reduces the complexity of the implementation at any one stage, but allows for the easy substitution of similar algorithms into any stage. For example, the Primitives algorithm which is used to determine the actual path around obstacles and can take dynamics such as minimum turning radii and/or other constraints into account, could be replaced by the MILP path planning methods proposed in [2.15] without altering the performance of the new G^*_{TA} and G^*MILP methods in stage 6.

2.3.1 RoboFlag Problem Definition Implementation

The input to the task allocation method(s) of stage 6 from stage 5 is standardized to be:

1. A highly connected directional graph where each vertex, $v_i = [x_i,y_i]$ is defined as either a source (a robot) or a target and each graph edge, $e_{ij}(v_i,v_j)$, is assigned a weight, $w_{i,j}$, equal to the cost of the path between vertices v_i and v_j.
2. A sorted list of all edges, E

In this paper, the symbol "V" is used to represent the set of all vertices, "S" is the subset of source vertices (the robots) and "T" the subset of target vertices, which contain "v", "s", and "t" members each respectively.

$$S \subset V, \quad T \subset V, \quad s.t. \forall V_i \in V \ \ if\,(V_i \in S) \Rightarrow (V_i \notin T) \tag{2.2}$$

$$v = s + t \tag{2.3}$$

The output was standardized as:

1. The final cost associated with the optimal solution, J^*. For this paper's tests, $J(Tr_i)$ was calculated as the distance traveled in trip Tr_i , where $i \in \{1..s\}$.
2. The trip that should be executed for each source, Tr_i, , $i \in \{1..s\}$.
3. A report on the constraints or resource usage. For this paper's tests, the constraint was the amount of fuel used.
4. The method's total computation time

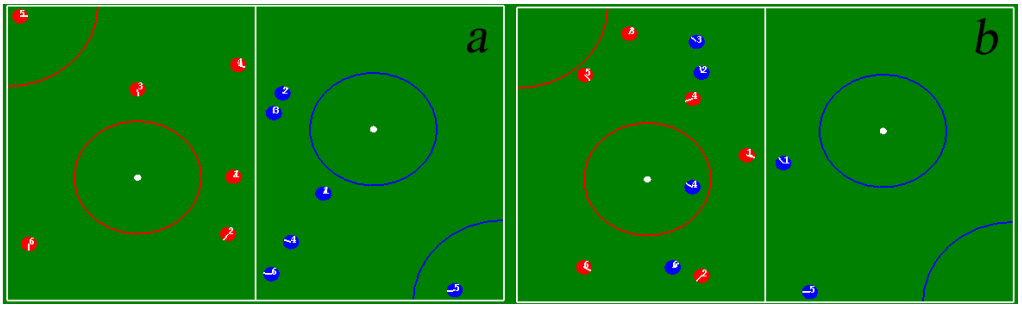

Figure 2.2: RoboFlag as a highly dynamic environment.

Less than 2 second difference between "a"&"b"

In all tests presented in this paper, the algorithms used at all stages, except for stage 6, were kept constant to keep any "background" influences consistent. Hence

they will not be discussed further in this paper. However, this overview is discussed to emphasize that all stages must be completed in under 0.5 seconds to be useful in real-time applications. As can be seen in Figure 2.1, the RoboFlag environment can change dramatically in a 1 second period of time. A highly conservative estimate of 30 ms has been used for the combined computation time of levels 1-4 & 7.

2.4 The G^*_{TA} Task Allocation Method

The G^*_{TA} task allocation method can best be described in terms of two separate components as depicted in Figure 2.3 below and explained in detail throughout Section 2.4. The primary component is built upon the A^* framework and will hence be referred to as A^*_{TA}. This part, described in Section 2.4.1 – 2.4.4, functions by growing partial solutions (or nodes) and evaluating these solutions based upon an optimistic predictive estimation cost until an optimal solution is grown and later identified. The operation of this A^*_{TA} component is enhanced by the secondary component, a greedy upper bound algorithm, referred to as G_{TA}, and described in Section 2.4.5 – 2.4.6. After every iteration of A^*_{TA}, G_{TA} uses the "best" of the nodes created by A^*_{TA}, which is referred to n_{min}, to generate upper bounds on the cost. Hence, the G_{TA} component is used to reduce the search space of the A^*_{TA} component and the combination of these two algorithms is described in Section 2.4.6.

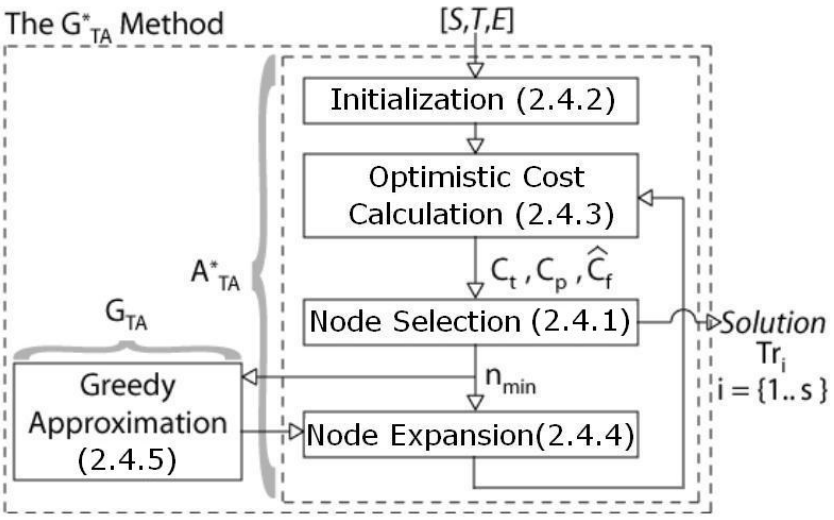

Figure 2.3: The G^*_{TA} Method Separated into it's A^*_{TA} and G_{TA} Components

2.4.1 The A Framework*

The casting of the A^*_{TA} component into the A^* framework can be easily described by first providing a few definitions. The most basic function of A^*_{TA} is the growth and evaluation of partial solutions. The most important characteristics of a partial solutions are stored as a "node" which is defined in (2.4) as 1) Tr_i, the trip created so far for source i, 2) R_{ij}, the usage of resource j by source i, 3) r, the number of resource constraints, and 4) U, the set of unassigned targets that are not yet part of any trip.

$$node := \begin{Bmatrix} Tr_{i,(partial\ solution)} \\ R_{i,j} \\ U \\ C_t, C_p, \hat{C}_f \end{Bmatrix} \quad \forall i \in \{1..s\},\ \forall j \in \{1..r\} \tag{2.4}$$

The last node parameters, the cost characteristics C_t, C_p, and $\hat{C}_f$, are most easily explained as a part of the A^*_{TA} general pseudo-code shown below which follows the 4 boxed steps of A^*_{TA} in Figure 2.3. This explanation also uses Figure 2.4 and Figure 2.5 to follow one single arbitrary node through the main loop, i.e. the loop of Figure 2.3 Boxes 2-5 where the details of Figure 2.3 Box 1 will be left for Section 2.4.2. In Figure 2.4 and 2.5, as is done throughout this paper, circles represent sources, squares represent targets and the edges connecting these shapes make up the node's sources' trips. However special to Figure 2.4, the thick lines will represent the prediction to complete a node's solution which, as will be explained in detail in Section 2.4.3, does not have to result in a feasible solution but rather only an optimistic one.

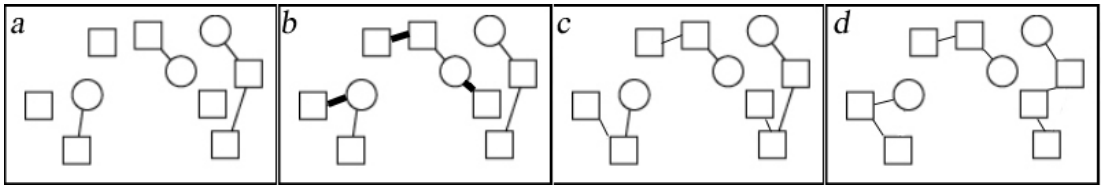

Figure 2.4: Circles=sources(*s*=3),Squares=targets(*t*=7),

a) an arbitrary node b) optimistic cost prediction for the node in a).

c) the best possible final solution starting from node a). d) the true optimal solution

1. Figure 2.3,Box 1: initialize A^*_{TA} with a new set of nodes from which all possible complete solutions could be grown from.
2. Figure 2.3,Box 2 : Calculate the cost to execute just the partial solution; the node's true cost, C_t. For the arbitrary node of Figure 2.4a, this is sum of the used edges' weights.
3. Figure 2.3,Box 2: create an optimistic predictive estimation of the best remaining cost that would be incurred in completing each new node's partial solution; the predictive cost, C_p. This is the sum of the thick edges in Figure 2.4b.

4. Figure 2.3,Box 3: store the new nodes in the master set of nodes, N, according to their combined true and predictive costs as shown in (2.5); the final cost estimation, $\hat{C}_f$
5. Figure 2.3,Box 3:choose the node from N with the lowest $\hat{C}_f$; n_{min}
   a. **if** this n_{min} is a complete solution, this node is the optimal solution
   b. **else** Figure 2.3,Box 4:delete n_{min} from N. Create a new set of nodes that all single step expanded versions of n_{min}'s partial solution, as demonstrated in Figure 2.5. Return to 2 with this new set of nodes.

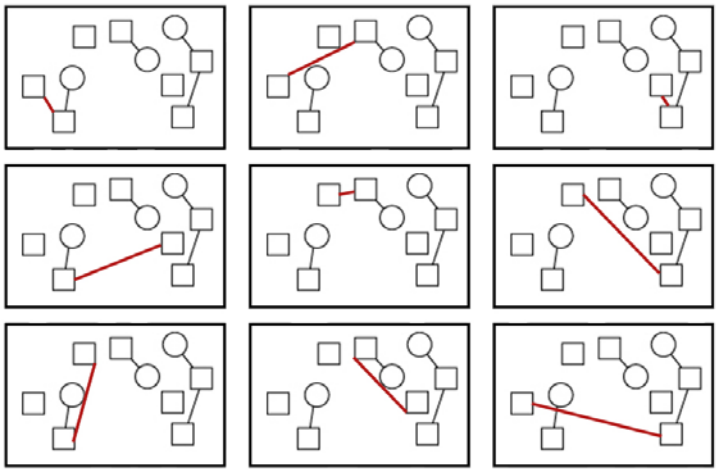

Figure 2.5: Circles=sources(s=3), Squares=targets(t=7),
All single step expansions of the arbitrary node of Figure 2.4a)
The red line in each box is the new single edge
added to the arbitrary node to form a new node

The A* framework's optimality guarantee has been proven several times and a simple example of a proof can be found in [2.36] and [2.37], however a short sketch of the proof is provided to aid the reader in identifying the significance of further Sections. In short, the guarantee that the complete solution from step 4a. is optimal stems from the A* framework's optimistic predictive estimation cost requirement. This

requirement states that given an arbitrary node, the estimate of the final cost for that node, $\hat{C}_f$, must always be less than or equal to the smallest final cost that could be obtained from starting with that partial solution, C_f^*. That is to say given the arbitrary node in Figure 2.4a, the estimate of its final solution Figure 2.4b must be less than or equal to what is the best final solution starting from the arbitrary node in Figure 2.4a. Henceforth the Figure 2.4b estimate is called "optimistic". This is also summarized in (2.5). Note however that C_f^* is not necessarily the global optimal cost, J^*, as the partial solution specified so far by an arbitrary node is not necessarily a part of the optimal complete solution, as is the case in shown in Figure 2.4a, 2.4c and 2.4d.

$$C_t + C_p = \hat{C}_f \leq C_f^* \tag{2.5}$$

For any complete solution, $\hat{C}_f$ equals C_f^* for that node as there is nothing to predict. Furthermore, in the case where n_{min} is a complete solution not only do all other nodes in N have an optimistic $\hat{C}_f$ no better than n_{min}'s (by n_{min}'s definition) but by (2.5) they must therefore have a best final cost C_f^* no better than n_{min}'s. This is summarized in (2.6).

$$C_f^{*}{}_{,n_{\min},complete} = \hat{C}_{f,n_{\min},complete} \leq \hat{C}_{f,n} \leq C_f^{*}{}_{,n} \quad \forall n \in N \tag{2.6}$$

This is obvious for any other complete solutions in N, but it is also true for any other partial solutions. Since all nodes are stored by their estimation $\hat{C}_f$ which is optimistic, expanding any other partial solution node at best could only result in a C_f^* equal to its $\hat{C}_f$ which again is already no better than n_{min}.

Remembering once again that all possible solutions could be created from the initial set of nodes, (as will be verified in Section 2.4.2), it follows for the ends of (2.6) that when n_{min} is a complete solution no other node could be expanded into a better solution and therefore n_{min} is the optimal solution. As will be shown in the sections below, the key to implementing the A^* framework in real-time then becomes being able to calculate C_t and C_p quickly and effectively so that $\hat{C}_f$ will be as close as possible to C_f^* while remaining optimistic. This concept is revisited in greater detail in Section 2.4.3.

2.4.2 Initialization

In the initialization stage the A^*_{TA} method does not yet look to eliminate possible solutions. Instead, the purpose of this stage is to create an initial set of nodes, N, from which all possible complete solutions could be grown from. The simplest set of nodes that could meet this criteria is the set that contains all possible partial solutions which pair exactly one target to exactly one source. For example, consider the simplest case of s=2 and t=2. As shown in the top of Figure 2.6, a set of four initial nodes are created by generating every possible assignment of only one target to only one source. This initial set which will always be of the size s x t, is essentially a set of all of the first steps that could be taken in creating a complete solution. From this initial set, every possible complete solution can be derived, as shown in the bottom of Figure 2.6.

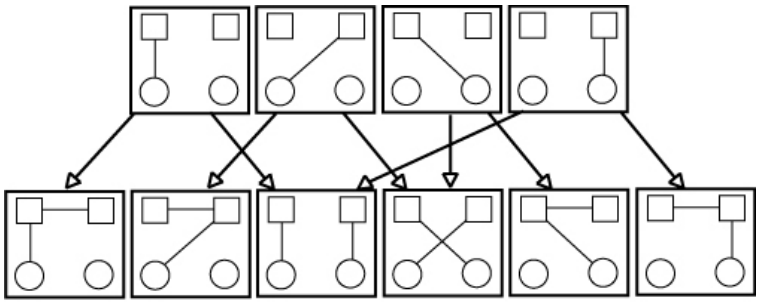

Figure 2.6: Node Initialization and Expansion

Circles=sources (s=2), Squares=targets (t=2)

As each node is created, the running costs C_t and C_p are calculated (by means shown in Section 2.4.3), R is updated, and the node is stored in by its final cost estimate, $\hat{C}_f$. Defining the operations below, the initialization pseudo-code can be written as follows:

n(·)	the **(·)** member of node n
n = MakeNode[n(Tr$_i$),v$_j$]	create a new node, n, that extends the end of trip Tr$_i$, to vertex v$_j$
CreateUnassigned[v$_j$,n(U)]	add all target vertices except for vertex v$_j$ to n(U).
CalcCost[n]	calculate C_t, C_p, and $\hat{C}_f$ of node n
Add[n,N]	add node n to the set N such that N will be sorted in ascending order by $\hat{C}_f$

```
1. N = {NULL}
2. for all source v_i ∈ S
3.         for all target v_j ∈ T
4.                 If e_ij(v_i,v_j) exists
5.                         n = MakeNode[n(Tr_i), v_j]
6.                         CreateUnassigned[v_j,n(U)]
7.                         CalcCost[n];
8.                         Add[n,N];
9. return N
```

Pseudo-Code 2.1: G^*_{TA} Initialization

It should be noted that constraints are not normally checked at the initialization phase because if the edge exists, a path exists between source v_i and target v_j, and the constraints were checked already in stage 5. If the constraints in stage 6 are different than those in stage 5, line 4 should be modified to include a constraint check as well.

2.4.3 Optimistic Predictive Cost Calculation

In A^*_{TA}, the true cost, C_t, is calculated as the sum of the costs of the chosen paths used in each source's trip, where again the paths' costs are provided in stage 5.

$$C_t = \sum_{i=1}^{s} (J_i(Tr_i)) \tag{2.7}$$

The predictive cost, C_p, however, is more complicated and is based on a minimum spanning forest (MSF) algorithm described here. The MSF is defined as a collection of *s* trees that each minimally span a subset of the entire set of vertices, *V*. The formation of this forest is constrained, however, in that each tree is required to contain exactly one source vertex, and each vertex (source or target) is exclusively a member of exactly one tree. This algorithm is explained first in pseudo-code and then later with the aid of an example in Figure 2.8.

The calculation of the MSF is based on concepts from Kruskal's minimum spanning tree (MST) algorithm [2.27]. Like Kruskal, the MSF algorithm looks to grow and combine trees by first assigning each vertex to its own unique set and set ID and then merges the vertex sets according to the smallest valid edges $\in E$.

In addition to this idea, the MSF algorithm also defines a subset Q as the list of set ID's equal to the sources' initial set IDs. In the MSF algorithm, vertex sets that both have a set ID $\in Q$ are not allowed to combine which ensures that no two sources will be a part of the same tree. Targets' initial sets, whose set ID $\notin Q$, can however combine with each other as well as with the sources' sets. If a target set, with set ID, j $\notin Q$, combines with a set with set ID, i $\in Q$, all vertices of the resulting merged set are assigned set ID, i $\in Q$.

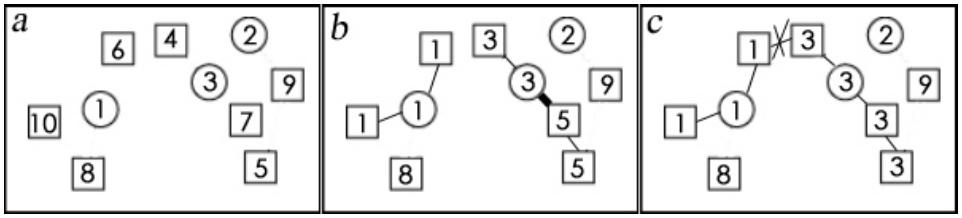

Figure 2.7: a) set IDs assigned to V, Q={1,2,3}

b) during MSF creation, the think line connecting 5 and 3 is allowed as $3 \in Q$, $5 \notin$ Q

c)later the connection between elements 1 and 3 is not allowed as both $1 \in Q$ and $3 \in Q$

As the process of combining sets continues, the number of sets will reduce to $|Q|$, (which also equals $|S|$). At this point, each set ID $\in Q$ corresponds to a different MST in the final MSF. The most important part of the algorithm is to keep track of the edges that are used to create the MSF as these edges will be used to later calculate the predictive cost, C_p. Defining a few simple operations, pseudo-code for the MSF creation is given as:

MakeSet[v_i]	assigns vertex v_i its own unique set ID.If $v_i \in S$, make the set ID of v_i part of Q
Set[v_i]	returns the set ID of vertex v_i
Union[v_i,v_j]	changes the set ID of all vertices of Set[v_i] to Set[v_j]
Store[e_{ij}, MSF]	make edge e_{ij} part of the set MSF, where MSF $\subset$ E
e_{ij}(v_i,v_j)	The edge, $e_{ij} \in E$, that connects vertex v_i to vertex v_j

```
1.  MSF = {NULL}, Q = {NULL}
2.  for all vertex v_k ∈ V, MakeSet[v_k]
3.  for all edge e_ij(v_i,v_j) ∈ E taken in ascending weight, W_ij, order
4.      if Set[v_i] != Set [v_j]
5           if (Set[v_i] ∈ Q) AND (Set[v_j] ∉ Q)
6.              Union[v_i,v_j]
7.              Store [e_ij, MSF]
8.          else if (Set[v_i] ∉ Q) AND (Set[v_j] ∈ Q)
9.              Union[v_j, v_i]
10.             Store [e_ij, MSF]
11.         else if (Set[v_i] ∉ Q) AND (Set[v_j] ∉ Q)
12.             Union[v_i,v_j]
13.             Store [e_ij, MSF]
14. return MSF
```

Pseudo-Code 2.2: MSF Creation for the G^*_{TA} Method

Figure 2.8 shows an example of an initial highly connected graph and the resulting MSF. Again notice that each tree is a MST for the subset of vertices included in that tree and that there are s=3 trees with each tree containing exactly one source.

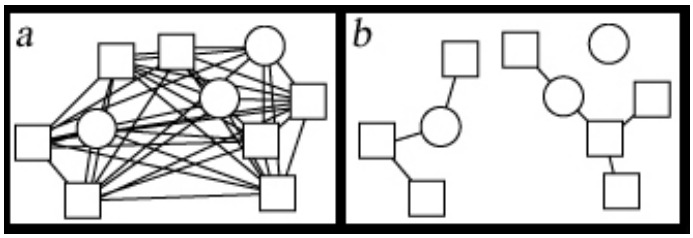

Figure 2.8: a)highly connected graph input b)resulting MSF

Circles=sources(*s*=3), Squares=targets(*t*=7)

With the MSF subset of edges defined, the optimistic predictive cost, C_p, for any partial solution can now be found. This process is best described with the use of an example and then the algorithm is formally defined below. The same MSF created from the graph in Figure 2.8b is shown again in Figure 2.9a. Figure 2.9b shows a potential partial solution that could be created from the same graph in Figure 2.8a. The predictive cost, C_p, for this partial solution is then found by connecting the unassigned targets of the partial solution to *any* part of the partial solution's trips using the smallest edges of the MSF subset as possible. It follows that the sum of the weights of the used MSF edges then equals C_p. This is shown in Figure 2.9c. Although the resulting graphs in Figure 2.9c are not a feasible solution on their own, the important point is that C_p is optimistic because the MSF subset contains the smallest edges possible to connect all of the sources and targets and hence there is no better way to assign targets to trips.

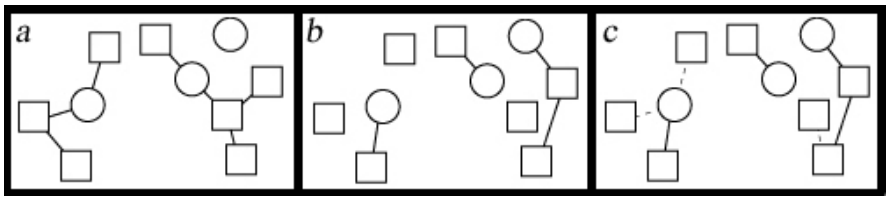

Figure 2.9: a) MSF b.) potential partial solution c.) edges used to calculate $\hat{C}_f$
Circles=sources (s=3), Squares=targets (t=7)

Creating the MSF edge subset, assuming that the total edge set E is sorted (as it should be provided from stage 5), is $O(|E|)$ which is nearly $O(|V|^2)$. The benefit of creating the MSF subset, however, is that the predictive cost, C_p, of any one partial solution can be calculated using only the $|V|$ MSF subset edges and hence this calculation will be only $O(|V|)$. This new algorithm which is based off of the original MSF creation algorithm is described in the pseudo-code below. In lines 3-5 all targets that are already part of a source's trip are immediately made a part of that source's set. This is done because the effect of assigning those targets to those sources' trips is already captured in the partial solution's true cost, C_t. Then starting at line 6, those targets not yet assigned to a trip are connected to the existing trips in a tree like fashion using the edges in the MSF subset. Also, since the MSF subset was built with the edges automatically entered in ascending weight order, no sorting is required in the C_p calculation.

```
1.  C_p = 0
2.  for all vertex v_k ∈ V, MakeSet[v_k]
3.  for all vertex v_i ∈ S
4.      for all vertex v_j ∈ Tr_i,
5.          Union[v_i,v_j]
6.  for all edge e_ij(v_i,v_j) ∈ MSF taken in ascending weight order
7.      if Set[v_i] != Set [v_j]
8.          if (Set[v_i] ∈ Q) AND (Set[v_j] ∉ Q)
9.          Union[v_i,v_j]
10.         C_p += w_i,j
11.     else if (Set[v_i] ∉ Q) AND (Set[v_j] ∈ Q)
12.          Union[v_j,v_i]
13.         C_p += w_i,j
14.     else if (Set[v_i] ∉ Q) AND (Set[v_j] ∉ Q)
15.         Union[v_i,v_j]
16.         C_p += w_i,j
17. return C_p
```

Pseudo-Code 2.3: Optimistic Predictive Cost Function of the G^*_{TA} Method

As will be discussed in later sections, in cases where there is a relatively high amount of branching, the effectiveness of the predictive cost, C_p, is lessened, which can increase the A^*_{TA} computation times. It should also be noted that direct use of this method should only be done if the cost for transitioning between targets is the same for all sources. If the cost is not the same, as may be the case for heterogeneous robots, the algorithm should be modified as discussed in Section 2.8.5.

2.4.4 Partial Solution Growth; Node Expansion

After creating an initial set of nodes, the A^*_{TA} component switches to the loop described below. This loop chooses one node at a time and creates new nodes based on the chosen node, n. Each new node extends one of the chosen node's trips by

exactly one of the chosen node's previously unassigned targets, $v_j \in n(U)$. This implies that there is a new node for every trip / single unassigned target combination. By examining each one of these possible new partial solutions, the algorithm ensures that all possible resulting complete solutions can be considered as again is required by the A^* framework. An example of the expansion of the arbitrary node of Figure 2.4a is shown previously in Figure 2.5. Defining the operations below, the node expansion pseudo-code can be written as follows:

Remove[v_j,n(U)]	remove element v_j from set n(U)
GetBest[N]	return the node,$n \in N$ with the best $\hat{C}_f$; and Remove[n,N]
ExpansionCheck[n,v_i,v_j]	if $e_{kj}(v_k,v_j)$ exists, where v_k is Tr_i's last vertex and the constraints check, return true
RepeatCheck[n,N,X]	examines sets N and X for a node with the same $\hat{C}_f$ as node n and the same partial solution. If a repeat node is found return false.

```
1.  K = {NULL}
2.  n = GetBest[N]
3.  while (n(U) != {NULL})
4.          Add[n,K]
5.          for all source vertex v_i ∈ S
6.                  for all target vertex v_j ∈ T
7.                      if ExpansionChecks[n,v_i,v_j]
8.                              n_new = MakeNode[n(Tr_i),v_j]
9.                              CalcCost[n];
10.                             if RepeatCheck[n_new,N,X]
11.                                     Remove[v_j, n_new(U)];
12.                                     Add[n_new,N]
13.                             else delete(n_new);
14.         n = GetBest[N]
```

Pseudo-Code 2.4 : G^*_{TA} Node Expansion

According to the A^* framework, if the node that is chosen to be expanded is always the node with the smallest final cost estimate $\hat{C}_f$, the first node to be chosen in line 2 or 14 to have n(U) = {NULL} will be the complete optimal solution.

2.4.5 G_{TA}: The GreedyUpper Bound Component of G^*_{TA}

As mentioned in [2.8], greedy approximation algorithms have been widely applied to the task allocation problem. The greedy algorithm shown in this section, G_{TA}, is a fairly simple algorithm that can be used to try to solve the entire task allocation problem defined in Section 2.2 very quickly. However as the full problem is NP-Hard [2.25], G_{TA} obviously cannot guarantee an optimal solution. In fact, because of the presence of constraints, it cannot even guarantee a complete solution. Despite this fact, G_{TA} will be shown to be a very useful component as the complete solutions that are generated by G_{TA} will be used to continually update an upper bound, $\hat{J}$, on the final global optimal cost, J^*.

$$\sum_{i=1}^{s} J\left((Tr_i)^*\right) = J^* \leq \hat{J} = \sum_{i=1}^{s} J\left((Tr_i)_{G_{TA}}\right) \tag{2.8}$$

The pseudo-code for the G_{TA} method by itself is given below.

SortEdges[v_k]	sort all outgoing edges of vertex, v_k
AddTarget[v_j,Tr_i]	add vertex v_j to the end of trip, Tr_i and remove it from U
[v_j,Tr_i]=GetSmallestEdge	returns Tr_i the trip with the smallest edge out of its last vertex to an unassigned target v_j.
ConstraintCheck[v_j,Tr_i]	if the constraints check for adding target v_j to trip Tr_i, return true
Remove[v_j,Tr_i]	remove from consideration the edge from Tr_i 's end to vertex v_j
ValidEdge[Tr]	returns true if there is an edge from the end of any trip, Tr, to an unassigned target.

```
1.  U = { all unassigned targets }
2.  for all vertex v_k ∈ V
3.      SortEdges[v_k]
4.  for all source vertex v_i ∈ S
5.      Tr_i = { v_i }
6.  while (U != {NULL} && ValidEdge[Tr])
7.      [v_j,Tr_i] = GetSmallestEdge
8.      if ( ConstraintCheck[v_j,Tr_i] )
9.          AddTarget[v_j,Tr_i]
10.     else
11.         Remove[v_j,Tr_i]
12. if (U != {NULL} ):  return solution cost as Ĵ
13. else:               return infinity
```

Pseudo-Code 2.5: G_{TA}, the Greedy Upper Bound Cost Estimate Component of the G^*_{TA} Method

In lines 2-3 the G_{TA} algorithm begins by creating a separate sorted list of the edges coming out of each vertex. In lines 4-5, the algorithm then initializes its solution with a set of *s* trips, one for each source, where each trip contains only that one source vertex. With these two elements, the G_{TA} algorithm enters into a while loop that adds

one unassigned target to the end of one of the trips in every complete iteration. This is done in a greedy fashion using the list of edges originating from the end vertex of each trip, $Tr_i(v_{end}(E))$. The smallest outgoing edge, $e_{min} \in v_{end}(E)$, that connects to an unassigned target is compared for each trip, and the trip with the smallest e_{min} is expanded to include that edge and the corresponding target. This target however is added *provided* that adding e_{min} and the target does not violate any of that source's trip constraints. If the constraints are violated, the next smallest outgoing edge, $e_{end,(\cdot)} \in v_{end}(E)$, connecting to an unassigned target for this trip is again compared with the other trip's smallest outgoing edges.

This "while loop" is continued until either 1) all of the targets are assigned to a trip or 2) all of the trips' outgoing edges, $Tr_i(v_{end}(E))$, are investigated and fail the constraint check process. In the first case, a complete solution is found and this solution's cost is returned as the new upper bound, $\hat{J}$. In the second case, no solution is found and a solution cost of infinity is returned.

2.4.6 G^*_{TA}: Combining A^*_{TA} and G_{TA}

The mainstay of combining the G^*_{TA} components is that the G_{TA} component's produced upper bound, $\hat{J}$, can be used to reduce the search space of the A^*_{TA} component. This is done by adding an additional check to the A^*_{TA} RepeatCheck function (described in Section 2.4.4) as follows. If a node's final cost estimate, $\hat{C}_f$ is greater than $\hat{J}$, then this node's partial solution does not need to be investigated any further because the G_{TA} solution has already been found to have a lower cost and still meets the constraints.

This final cost upper bound, $\hat{J}$, can also be updated throughout the execution of the A^*_{TA} component. As A^*_{TA} selects its best node to expand, n_{min}, G_{TA} can be run again using n_{min} as the input. Replacing lines 1-5 of the G_{TA} pseudo-code (as shown in Section 2.4.5) with the code below allows n_{min} to essentially seed G_{TA} with its partial solution.

```
1.  U = n_min(U)
2.  for all source vertex v_i ∈ S
3.      Tr_i = n_min (Tr_i)
```

Pseudo-Code 2.6: Modification of the GTA method to create a Complete Solution from an Input Partial Solution Node, n_{min}

Then the G_{TA} algorithm is asked to complete the solution the best that it can. If the resulting cost from this run of G_{TA} is better than the current final cost upper bound $\hat{J}$, the current $\hat{J}$ is replaced with this new solution cost and thereby narrows the search space even further.

Note that in the $\hat{J}$ update, the G_{TA} algorithm sort appears to be excluded. This is because the sort has already been done in the first G_{TA} run and therefore can simply be reused. Furthermore, since the sort was done for the edges of each vertex separately, not all of the edges need to be reconsidered. Instead only those at the end of the n_{min}'s trips and for the unassigned targets, $n_{min}(U)$ must be considered. This helps to reduce the computation time of the later G_{TA} calls significantly.

It is noted that using a separate sorted edge list for each vertex actually increases the "big O" running time of the G_{TA} algorithm over using a single sorted list

of all edges. However, for cases where the number of sources, s, and unassigned targets, $|U|$, are small enough, this method actually produces faster results, particularly because the vast majority of the G_{TA} calls are with $|U| < t$ and often $|U| << t$.

If the G_{TA} component never produces a solution, the A^*_{TA} component will still run and produce an optimal answer. However, as will be shown later in Sections 2.6 and 2.7, the extra effort spent on running G_{TA} is worth the investment.

2.5 MILP Based Methods

2.5.1 A Standard Approach, $MILP_{TA}$

As MILP is one of the most predominate methods for solving task allocation methods optimally [2.2],[2.7],[2.9]-[2.16],[2.22], several methods using MILP were developed in the RoboFlag testbed where the ultimate solutions were found using commercially optimized AMPL software as compared to the directly C++ coded software used for G^*_{TA}. The first method, $MILP_{TA}$, is a standard approach largely based on MILP techniques from [2.22]. In this case, stage 6 of the RoboFlag testbed was used only to create the necessary AMPL data files.

The primary component of $MILP_{TA}$, is the creation of a matrix, M, that represents all of the possible ordered combinations of the targets (trips) as well as the potential assignment of these trips to the sources. To create M, first let each row represent one target and let each column represent one potential trip. With this assignment the size of M can then be calculated as t x b where b is the sum of the number of possible permutations for the target set T times s to represent the possibility that any ordered arrangement of targets could be assigned to any source, as shown in (2.9).

$$b = \left(\sum_{k=1}^{t} \frac{t!}{(t-k)!} \right) s \tag{2.9}$$

In this matrix, M, let each entry in every column be the order in which that target is visited. So if there are 3 targets and the potential trip is to visit target 1 second, target 2 not at all, and target 3 first, the corresponding column would be [2, 0, 1]^{T}. Using this concept, the simple case of having only 1 source and 3 targets would then look like

$$Given \begin{pmatrix} s=2 \\ t=3 \end{pmatrix}, M = \begin{bmatrix} 1001\,122001\,12233 \\ 0102010\,1\,2231312 \\ 0010201\,2\,1333121 \end{bmatrix} \tag{2.10}$$

In order to now represent multiple sources, the matrix above is simply duplicated *s*-1 times and each matrix is appended to the rest to create the total M matrix as is shown in (2.11) for the two source case.

$$Given \begin{pmatrix} s=2 \\ t=3 \end{pmatrix}, M = \begin{bmatrix} \overbrace{1001\,122001\,12233}^{\text{source 1}} & \overbrace{1001\,122001\,12233}^{\text{source 2}} \\ 0102010\,12231312 & 0102010\,12231312 \\ 0010201\,21333121 & 0010201\,21333121 \end{bmatrix} \tag{2.11}$$

As every column is a potential trip, the cost for executing that trip for the respective source can easily be calculated as the summation of path costs used in that trip, as provided from the path information from RoboFlag stage 5. These potential trip costs are then stored together in a cost column vector, C_v of length *b*. Similarly,

whether a potential trip is selected to be a part of the final solution or not can be stored in a binary variable row vector, A[b] (2.12). Using these two vectors, the cost equation can be easily written as (2.13).

$$A[i] = \{0,1\} \quad \forall i \in \{1..p\} \tag{2.12}$$

$$minimize\left(\sum_{j=1}^{p} A[j]C_v[j]\right) \tag{2.13}$$

The more challenging part however is to enforce that every target is assigned to exactly one source. To do this, M must first be transformed into a binary matrix where every entry that is nonzero is set to one and every zero entry remains as zero (2.14).

$$Given\begin{pmatrix} s=2 \\ t=3 \end{pmatrix}, M = \begin{bmatrix} \overbrace{1\,0\,0\,1\;1\,1\,1\,0\,0\,1\,1\,1\,1\,1}^{\text{source 1}} & \overbrace{1\,0\,0\,1\;1\,1\,1\,0\,0\,1\,1\,1\,1\,1}^{\text{source 2}} \\ 0\,1\,0\,1\,0\,1\,0\,1\,1\;1\,1\,1\,1\,1 & 0\,1\,0\,1\,0\,1\,0\,1\,1\;1\,1\,1\,1\,1 \\ 0\,0\,1\,0\,1\,0\,1\,1\,1\;1\,1\,1\,1\,1 & 0\,0\,1\,0\,1\,0\,1\,1\,1\;1\,1\,1\,1\,1 \end{bmatrix} \tag{2.14}$$

$$\left(\sum_{j=1}^{p} A[j]M[i,j]\right) = 1, \quad \forall i \in \{1..t\} \tag{2.15}$$

$$\left(\sum_{j=(i*p)+1}^{p*(i+1)} A[j]\right) \le 1, \quad \forall i \in \{0..(s-1)\} \tag{2.16}$$

This transformed M now simply states that in every trip column where there is a "1" in row j, the j^{th} target is a part of this trip. In order to ensure that every target is part of only one of the assigned trips, the sum of every row of the selected subset of columns that make up a solution must exactly equal one. This constraint is represented in (2.15) where M[i,j] represents the i^{th} row, j^{th} column element of a given M binary matrix. Similarly, the constraint that each source can only be assigned one trip, is the

same as requiring the final solution to only contain no more than one column from each repetition of the source sub-matrix (2.16). (Note: although (2.14) may appear to have duplicate columns, each one of these columns has a unique cost associated with it as is calculated via (2.10,) and (2.12) and stored in C_v.)

2.5.2 G^*_{TA} / MILP Combination Methods, G^*MILP_{TA}

Although the $MILP_{TA}$ method can be represented concisely with the equations shown above, it is easy to see that equation (2.9) and hence M grow very quickly as *s* and *t* increase. To address this issue, a new method that combines G^*_{TA} with MILP was also created; this is referred to as G^*MILP.

The major difference in G^*MILP is in the creation of the M matrix. In this method, the M matrix only exists as a binary matrix where the rows still represent targets but each of the columns can now be thought of as a solved TSP problem. For example, consider the fifth row of the M matrix in (2.17): $[1\ 0\ 1]^T$.

$$Given\begin{pmatrix} s=2 \\ t=3 \end{pmatrix}, \quad M_{TSP} = \begin{bmatrix} \overbrace{1001101}^{\text{source 1}} & \overbrace{1001101}^{\text{source 2}} \\ 0101011 & 0101011 \\ 0010111 & 0010111 \\ \end{bmatrix} \tag{2.17}$$

Similar to the $MILP_{TA}$ case, this column states that the first and third targets are a part of this potential source 1 trip. The difference in G^*MILP however is the cost associated with this column, C_v. Unlike in $MILP_{TA}$ where there was an integer M counterpart that specified the order to visit the targets, the order the targets are visited in G^*MILP is decided by solving a TSP problem using source 1 and targets 1 and 3 as

inputs to G^*_{TA}. This greatly reduces the number of columns in M as can be seen in even this simple case. In (2.17), G^*MILP has only one column for targets 1 and three in (2.11) and (2.14) $MILP_{TA}$ had two columns: one for visiting target 1 first and target 3 second and another for target 3 first and target 1 second.

As *s* and *t* increase, the column reduction is improved exponentially and the number of possible trips in M_{TSP} is equal to the sum of the number of possible combinations of the target set *T*, times the number of sources, *s* (2.18).

$$b_{TSP} = \left(\sum_{k=1}^{t} \frac{t!}{k!((t-k)!)} \right) s \tag{2.18}$$

Although this comes at an extra computation cost of solving the TSP problems, many of them are fairly small scale; as will be seen in Section 2.6 this is a worthwhile investment. Aside from this change in M to M_{TSP} and the value of *b* to b_{TSP}, the rest of the MILP setup remains the same and equations (2.12), (2.13), (2.15) and (2.16) still apply and are used to complete G^*MILP.

2.6 Implementation Test Results

As the methods presented here vary greatly, it is hard to determine from the descriptions alone which will produce the best results. However, from the test results presented in this section it is very clear to see the benefits that can result from applying these new methods. The methods compared in this section are G^*_{TA}, $MILP_{TA}$, and G^*MILP, as well as a method that only ran the A^*_{TA} component of G^*_{TA} which will simply be referred to as A^*_{TA}, and a method that is a version of G^*MILP that only ran the A^*_{TA} component of G^*_{TA} to solve the G^*MILP TSP problems, which will be

referred to as A^*MILP. As a reminder, each of these methods always found identical optimal solutions. Thus, the key comparison is the computation times.

All tests were performed on the Cornell RoboFlag testbed using a 2.0GHz dual processor PC with 1 GB RAM. The tests were setup to vary the number of sources s = {2..9}. The number of targets, t, for these tests could only be varied from t = {2..6} due to the fact that if t was increased beyond 6, the MILP based methods required more memory than was available on the test PC. The G^*_{TA} method and the A^*_{TA} only method however, did not require as much memory. Therefore the tests for these methods in this section were be carried out to s=9 sources and t=9 targets with system memory to spare. For each possible (s,t) pair, 100 tests of randomly generated source and target position sets were created across the RoboFlag field. The average computation time for selected test sets are displayed in the table and graphs below. In particular, as many of the RoboFlag based experiments focus on a maximum of 6 robots per team, the s=6, t=6 case is highlighted and will be used as a benchmark for comparison.

Table 2.2: Average Computation Time Comparison of the A^*_{TA}, G^*_{TA}, $MILP_{TA}$, A^*MILP, and G^*MILP Methods in milliseconds.

s	*t*	A^*_{TA}	G^*_{TA}	$MILP_{TA}$	A^*MILP	G^*MILP
2	2	0.16	0.16	209.71	201.76	202.16
2	4	0.67	0.66	221.05	209.27	210.46
2	6	3.85	3.14	813.51	254.65	263.15
2	9	336.67	104.01			
4	2	0.32	0.29	205.88	203.49	203.52
4	4	1.69	1.21	263.86	244.09	246.73
4	6	37.96	15.88	1,507.03	341.20	358.17
4	9	9,919.27	1,113.73			
6	2	0.53	0.45	208.73	205.31	206.26
6	4	2.84	1.80	252.37	224.56	227.58
6	**6**	**98.33**	**29.01**	**2,201.20**	**362.25**	**388.24**
6	9	43,830.60	4,155.66			
9	2	0.94	0.77	209.83	207.39	207.62
9	4	5.16	2.83	276.32	236.87	240.84
9	6	265.19	72.92	3,336.61	445.54	483.68
9	9	225,896.00	13,934.30			

One of the most remarkable results is that for all (*s,t*) pairs, the G^*_{TA} method ran two orders of magnitude faster than $MILP_{TA}$. The A^*_{TA} only method also did well but was never as fast as G^*_{TA} hence demonstrating the value of adding the G_{TA} component. The reason for the significant difference in times is largely due to the fast upper bound estimates created by the G_{TA} component which in turn greatly reduces the number of nodes that must be created. Most importantly however is not the time saved in node creation, but rather the time saved from the reduction in the number of nodes that had to be stored in a sorted data structure.

The G^*MILP and A^*MILP methods also did well, running anywhere from roughly in the same amount of time as MILP for the smallest scale cases to almost an order of magnitude faster in the larger scale cases. These results clearly indicate that

the new combined MILP methods show potential for improved scalability as compared to $MILP_{TA}$. Interestingly, the A^*MILP method in several cases did slightly better than G^*MILP. In these cases, it appears that the added benefit of using G_{TA} was countered by the extra overhead required to run it. This difference is due to the fact that many of the TSP problems that need to be solved in G^*MILP are very small in comparison to the problems it is asked to address in the G^*_{TA} application.

For completeness, the computational time standard deviations were also examined. This analysis provided little additional insight but the results are quickly summarized here. In the MILP based methods tests, the standard deviations are consistently one order of magnitude less than their averages, but for the G^*_{TA} method, its standard deviations are almost equal to its averages. In some applications this could cause potential concern for the G^*_{TA} method. However, for the benchmark case, as the G^*_{TA} method is so fast on its own, the G^*_{TA} averages and standard deviations are low enough not be of a consequence. In addition, even if the G^*_{TA} averages are increased by *two* standard deviations, in all cases the adjusted averages would still be at least nearly half the A^*MILP and G^*MILP average times and at least one order if not two orders of magnitude faster than $MILP_{TA}$. Also in Section 2.7 an approach for dealing with larger scale cases is addressed.

As this is a NP-Hard problem [2.25], it is not surprising that all methods grow exponentially, however what is interesting to note is that the methods grow at a slower rate as *s* increases than as *t* increases. This can be seen from the equations in earlier sections but is also easily verified by comparing the semi-log plots of Figure 2.10 thru 2.16. These figures were created for the varying values of *s* and *t* up to s=6, t=6, which again represents the maximum size problems the MILP based methods could handle

on the test PC. In Figure 2.10 thru 2.13 it is clear G^*_{TA} is by far the fastest and it scales just as well as A*MILP and G*MILP in the larger scale cases. In Figure 2.14 thru 2.16 however, although it is clear that G^*_{TA} is once again the fastest, it is not clear which method scales better as *t* increases.

As is shown by the results expressed in Table 2, it appears that there is an initial computation time threshold for all of the MILP based methods after which the methods begin to grow at a faster rate. Examining the largest scale case of s=6, t={2..6} as can be seen in Figure 2.15, the $MILP_{TA}$ method begins to grow very quickly between the $t = 4$ and $t = 5$ point and the exponential growth of the method is seen very clearly as it advances to t=6 point. In Figure 2.15, it also appears that A*MILP and G*MILP begin to grow by the t=6 point. However, the growth rate of the next *t* increase could not be experimentally determined because of the limitations imposed by the MILP method's great memory usage.

Regardless of these G^*_{TA} and G*MILP comparisons, it is certain that all new methods provided in this paper are computationally faster than $MILP_{TA}$. Also for the applications proposed in this paper, the G^*_{TA} method can provide *optimal* solutions for the benchmark s=6, t=6 case and be within the desired runtime of less than 0.5 sec.

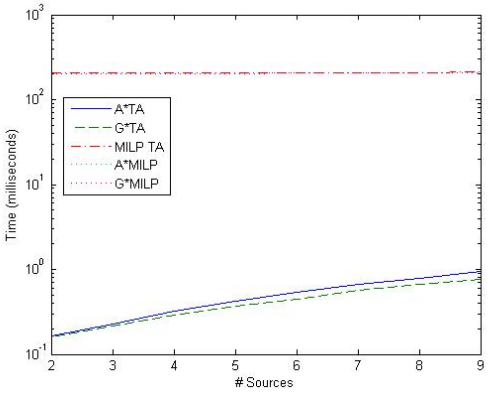

Figure 2.10: Method Time Comparison s={2..9}, t=2

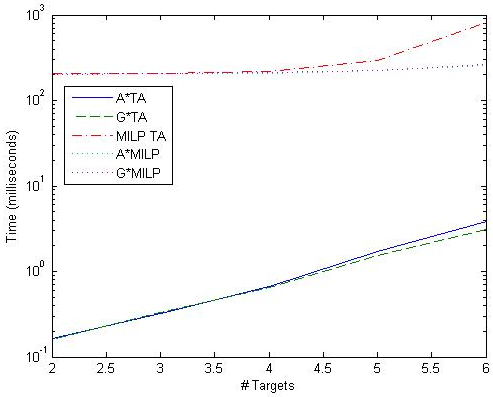

Figure 2.13: Method Time Comparison, s=2, t={2..6}

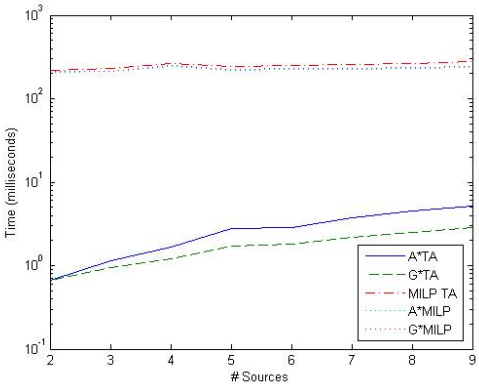

Figure 2.11: Method Time Comparison, s={2..9}, t=4

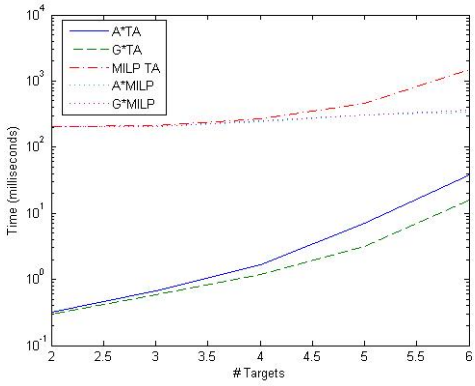

Figure 2.14: Method Time Comparison, s=4, t={2..6}

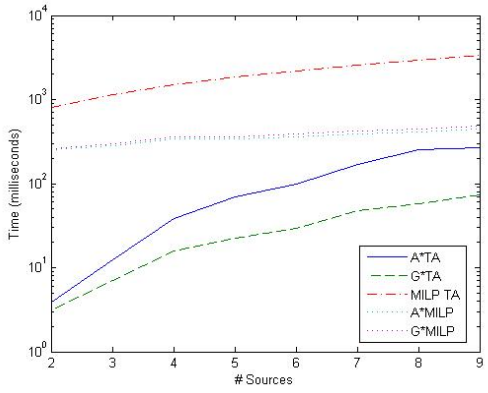

Figure 2.12: Method Time Comparison, s={2..9}, t=6

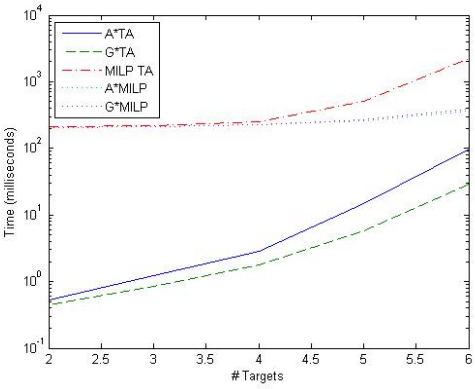

Figure 2.15: Method Time Comparison, s=6, t={2..6}

The speed and optimality guarantee that G^*_{TA} provides make it an excellent method for a variety of potential applications. However, there are applications which require solutions in faster times than even those presented here. For this reason, the G^*_{TA} method was designed to be able to provide an approximation of the optimal solution at almost any point during its execution by simply accessing the answer provided by the G_{TA} component.

2.7 G^*_{TA} Strict Time Requirements

In this section, the results of a set of tests similar to the ones performed in Section 2.6 are presented. For these tests, 100 randomly generated data sets were created for all possible (*s,t*) pairs with *s*={2..9} and *t*={2..9}. In these set of tests, the full G^*_{TA} method was applied along with runs of the full G^*_{TA} method that were stopped at times of 0.025, 0.1, and 0.5 seconds. The results of these tests are shown below in Table 2.3.

Table 2.3: Average Time to Complete Optimal G^*_{TA} (milliseconds) vs. Percent error (%E) in Time Limited G^*_{TA} Runs

s	*t*	G^*_{TA} (time, ms)	%E, G^*_{TA} (0.025 s)	%E, G^*_{TA} (0.1 s)	%E, G^*_{TA} (0.5 s)
2	2	0.05	0.00	0.00	0.00
2	4	0.28	0.00	0.00	0.00
2	6	2.47	0.00	0.00	0.00
2	9	134.36	0.61	0.08	0.00
4	2	0.06	0.00	0.00	0.00
4	4	0.71	0.00	0.00	0.00
4	6	16.47	0.00	0.00	0.00
4	9	1,512.69	1.13	0.45	0.35
6	2	0.07	0.00	0.00	0.00
6	4	1.28	0.00	0.00	0.00
6	6	38.57	0.08	0.00	0.00
6	9	3,425.28	1.46	1.14	0.88
9	2	0.11	0.00	0.00	0.00
9	4	1.84	0.00	0.00	0.00
9	6	118.34	0.04	0.00	0.00
9	9	14,665.00	0.64	0.47	0.45

As can be seen, the time limited G^*_{TA} methods produce results that are very near optimal. In the 0.025 second case, which is below even the 30 Hz RoboFlag system rate, the average percent error never goes above 2%. When the time is increased to 0.5 seconds the average drops to less then 1%. From these results it can be seen that near optimal solutions can be obtained very quickly and that there is typically a very small gain on average for the extra computational time invested.

By itself, this would make solving the full problem on its own not seem worthwhile for many applications. However, the percent error occasionally spikes which can be cause for some concern. The largest percent error spike that was found in all runs was nearly 20%. These spikes appear to occur when it is difficult to

determine what are the best initial steps due to the costs being very close to one another. However as the averages are so low, this rarely happens. Therefore, in applications that handle occasional spikes, this time limited variation of G^*_{TA} can offer a very powerful new method.

2.8 Task Allocation Problem Variations

One of the advantages of the G^*_{TA} is that the algorithms presented here can be easily modified to handle variations of the task allocation problem. This section provides both descriptions on how to modify the previously presented algorithms for these variations as well as observations on the resulting effects and trends.

2.8.1 Required End Vertices

It may be desired that the trips of the vehicles end at certain vertices. An operation can be added between lines 5 and 6 in the node expansion algorithm that states if this source's trip ends in a desired end vertex, do not extend that trip any further and therefore continue to the next iteration of the line 5 for loop. This has the effect of reducing the number of possible new nodes that are created from an expansion and can therefore reduced the computation time.

2.8.2 Source Specific Targets

This variation handles the case when it is desired that only certain vehicles be allowed to visit certain targets. For this case, the ExpansionCheck of line 7 of the node expansion algorithm performs an extra check to see if the given v_i source, and v_j target are an acceptable match. If they are not a match, the algorithm simply returns back to line 6 and continues. This check can be implemented with the aid of storing the acceptable subset of the targets for each source or the acceptable subset of the sources

for each target. Again this reduces the number of new nodes created from a node expansion and can therefore reduce the overall computation time.

In the supported G*MILP case this is most easily handled by not solving the TSP problems where the specific target and source are not a match. Instead the corresponding column is simply eliminated from the TSP matrix. Hence the problem becomes smaller instead of having to add additional constraints.

2.8.3 Round Trip

This variation of the problem requires that the vehicles return to their original state at the end of their trip. It can be thought of as a special case of the required end vertices and source specific variations where an additional target vertex is added to V for every source, that is identical to the corresponding source vertex. These additional target vertices are then made source specific to the corresponding source. In general this increases the computation time of the MILP methods as the TSP problems have all been increased by one target and the overall problem has been increased by one target per source. Typically though the computation times are not as large as compared to simply adding one vertex per source as some of the benefit from the variations discussed above is gained.

2.8.4 Target Priority

In the problem described in Section 2.2, all targets are treated as having the same priority. However, the input to stage 6 can be changed easily to provide a priority scheme. To begin, each target ranging from the topmost priority to the bottom most priority is assigned a priority value ranging between one and a maximum value respectively. These values then act as edge weight scaling factors. Every edge coming

into a target is multiplied by their priority scaling value and hence the cost to visit bottom priority targets is increased and those of the topmost priority remain the same. The only additional computation time that results from implementing priorities is the time required to the assign the priorities which is O(*E*). However, as the process is very straightforward in most cases, this extra time is essentially negligible as it is usually far less than even a tenth of a millisecond.

2.8.5 Heterogeneous Robots

In the case where the vehicles have significantly different characteristics and constraints, the costs between targets will not necessarily be the same. Therefore, the G^*_{TA}'s cost prediction step must be modified. Instead a separate MSF must be created for each vehicle type and in the limiting case where all vehicles are different, a total of *s* MST's must be created. With this added complexity the predictive cost algorithm must be modified to resemble Prim's MST algorithm more than Kruskal's. In lines 7,10 and between lines 12 and 13, of the MSF algorithm, a modification must be made such that as each MSF or MST is created, the chosen edges for each vertex *v* are stored in a separate sorted edge set, Pr_v, specific to each vertex. Allowing the Union[X,Y] operation to make set X a part of set Y as well, the C_p, predictive cost, calculation algorithm then can be described as:

```
1.  Cp = 0, G = {NULL}
2.  for all vertex vk ∈ V, MakeSet[vk]
3.  for all vertex vi ∈ S
4.      for all vertex vj ∈ Tri,
5.          Union[vi,vj]
6.          Union [Prj,G] keeping G sorted
7.  for all edge eij(vi,vj) ∈ G taken in ascending weight order
8.      if Set[vi] != Set [vj]
9       .      if (Set[vi] ∈ Q) AND (Set[vj] ∉ Q)
10.                 Union[vi,vj]
11.                 Union [Prj,G] keeping G sorted
12.                 Cp += wi,j
13.         else if (Set[vi] ∉ Q) AND (Set[vj] ∈ Q)
14.                 Union[vj,vi]
15.                 Union [Pri,G] keeping G sorted
16.                 Cp += wi,j
17. return Cp
```

Pseudo-Code 2.7 : Modification to the MSF Algorithm to Account for Heterogeneous Sources

In the limiting case this increases the predictive cost algorithm component's order from ($|V|$) to ($s|V|$) but it does not increase the number of nodes created which is the largest factor for increasing computation time. Keeping the G set sorted can also add to the order but as the sorted order of potential edge members can be established in the initial MSF creation step, this addition can be eliminated. This new algorithm does not increase the stage 6 computation time of the G*MILP method as only one source is considered during any TSP problem. However, in both methods the stage 5 primitives computation time could increase by up to a factor of s. However stage 5

times have been shown to always be relatively small, i.e. on the order of tenths of milliseconds.

2.8.6 Constraints

Many times in implementing MILP methods, adding constraints generally increases computation times. [2.8],[2.22] In G^*_{TA} however, adding constraints typically decreases the computation times. This can be explained in that constraints can prevent the creation of new nodes during the node expansion step and hence in effect limit or rather reduce the search space. Out of the task allocation tests a direct correlation showing that the larger the numbers of nodes the larger the computation time was seen in every test and in developing the G^*_{TA} adding algorithmic features like the G_{TA} component to reduce the number of nodes had the greatest effect in reducing computation time.

2.8.7 Minimizing Individual Cost

The problem as defined in Section 2.2, has been setup to minimize the total cost "spent" by all vehicles. In some cases it may be more preferable to spread the cost out as much as possible and minimize the cost that any one vehicle must spend. To handle this in G^*_{TA}, C_t and C_p are calculated in the same way, however, the equation for $\hat{C}_f$ is changed and a new member is added to the node data structure. This new member tracks the current highest cost of any of the node's trips, $C_{Tr,max}$. With C_{Tri} as the cost of source i's trip, $\hat{C}_f$ is then calculated as follows.

```
1. for each Tr_i ∈ node n, where i ∈ S
2.        C_p -= (C_trmax - C_tri)
3. if C_p > 0
4.        Ĉ_f = C_t + C_p
5. else Ĉ_f = C_t
```

Pseudo-Code 2.8: Modification to the G^*_{TA} Cost Equation to Optimize the Minimal Individual Cost of Any One Sources

This change maintains the A^* requirement that $\hat{C}_f$ of a node always be optimistic and therefore optimality is guaranteed. However, it does not provide as good of an estimate of which node may lead to the optimal answer and therefore solving this variation can lead to relatively higher computation times. Similarly, the G_{TA} component also needs to be modified so that the cost to minimize is not the sum of the trip's costs but rather the maximum cost of any one trip.

The computation times for the supported MILP method increase as well as the number of variables must increase to store the maximum cost of any one TSP. Similarly, the new cost equation to minimize is:

$$minimize\left(\max\left(A[j]C_v[j]\right)\right), \quad \forall j \in \{1..p\} \tag{2.19}$$

2.9 Conclusions

From the tests presented, a new task allocation method G^*_{TA} has been established as a viable and fast technique for solving challenging task allocation problems optimally in real-time. The results indicate at least a two order of magnitude drop in computation time as compared to a standard implementation of MILP. In addition, G^*_{TA} has also been shown to be effective in combining with MILP to create

G^*MILP which has shown promising scaling potential. Finally, the G^*_{TA} algorithm has also been shown to be able to provide very near-optimal solutions in highly time restricted applications.

REFERENCES

[1] Campbell, M., "Planning Algorithm for Multiple Satellite Clusters," *Journal of Guidance, Control and Dynamics*, Sept-Oct 2003.

[2] G. Thomas, A. M. Howard, A. B. Williams, and A. Moore-Alston, "Multi-robot task allocation in lunar mission construction scenarios," in *IEEE International Conference on Systems,* Oct 2005

[3] Chandler, P., Pachter, M., *et al.* "Distributed Control for Multiple UAVs with Strongly Coupled Tasks," AIAA *Guidance, Navigation, and Control Conference*, August 2003

[4] Bellingham, J., Tillerson, T., Richards, A., How, J., "Multi-Task Allocation and Path Planning For Cooperating UAVs" *Conference on Coordination, Control and Optimization*, Nov. 2001

[5] Purwin O., D'Andrea R.: "Cornell Big Red 2003", in: Polani D., Bonarini A., Browning B., Yoshida K. (Eds), *Robocup 2003: Robot Soccer World CupVII, Lecture Notes in Artificial Intelligence,* Springer, Berlin, 2003

[6] D. Dyke Weatherington, "DoD UAV Roadmap", *U.S. Department of Defense*, 2003.

[7] Richards A, Bellingham J, Tillerson M, and How J, "Coordination and Control of Multiple UAVs", *Guidance Navigation and Control Conference*, Aug. 2002.

[8] Gerkey, B., Mataric, M., "A formal analysis and taxonomy of task allocation in multi-robot systems" *International. Journal of Robotics Research* 23(9):939-954, September 2004

[9] Y. Kuwata, A. Richards, T. Schouwenaars, and J.How, "Distributed Robust Receding Horizon Control for Multi-vehicle Guidance" *IEEE Transactions on Control Systems Technology Journal*, accepted 2007

[10] C. Schumacher, P. Chandler, M. Pachter, and L. Pachter, "UAV Task Assignment with Timing Constraints via Mixed-Integer Linear Programming", *AIAA 3rd "Unmanned Unlimited" Technical Conference, Workshop and Exhibit*, September 2004

[11] M. A. Darrah, W. Niland, and B.M.Stolarik, "Multiple UAV Dynamic Task Allocation Using Mixed Integer Linear Programming in a Sead Mission," in *Infotech@Aerospace*, Arlington, Virginia, September pp.26-29, 2005.

[12] N. Atay, and B.Bayazit "Mixed-Integer Linear Programming Solution to Multi-Robot Task Allocation Problem" *IEEE International Conference on Robotics and Automation*, 2007

[13] A. Bender. MILP based task mapping for heterogeneous multiprocessor ystem" *In Proceedings of EURO-DAC*, September 1996.

[14] A.Davare, J.Chong, Q. Zhu, D. Densmore, A. Sangiovanni-Vincentelli, "Classification, Customization, and Characterization:Using MILP for Task Allocation and Scheduling" University of California Berkley, Technical Report No. UCB/EECS-2006-166, December 2006.

[15] M. G. Earl and R. D'Andrea, "Iterative MILP Methods for Vehicle Control Problems," *IEEE Conference on Decision and Control*, Atlantis, Paradise Island, Bahamas, Dec. 2004

[16] Richards, A., Kuwata, Y., How, J., "Experimental Demonstrations of Real Time MILP Control" *AIAA Guidance Navigation and Control Conference*, Aug. 2003.

[17] P. B. Sujit, A. Sinha, and D. Ghose, "Multi-uav task allocation using team theory," in *IEEE International Conference on Decision and Control, and the European Control Conference*, Seville, Spain, December 12-15 2005, pp. 1497–1502.

[18] R. Zlot and A. Stentz, "Complex task allocation for multiple robots," in *International Conference on Robotics and Automation*, Barcelona, Spain, April 2005, pp. 1515–1522.

[19] B. P. Gerkey and M. J. Matari´c, "Sold!: Auction methods for multirobot coordination," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 5, pp. 758–786, October 2002.

[20] R. Zlot, A. Stentz, M.B. Dias, and S. Thayer, "Multi-robot Exploration Controlled by a Market Economy" *IEEE International Conference on Robotics and Automation*, 2002

[21] T. Lemaire, R. Alami, and S. Lacroix, "A distributed tasks allocation scheme in multi-uav context," in *IEEE International Conference on Robotics and Automation*, New Orleans, LA, April 2004, pp. 3822–3827.

[22] Ousingsawat, J., and Campbell, M., "Multiple Vehicle Team Tasking for Cooperative Estimation", *American Control Conference*, 2004

[23] S. Rathinam, and R. Sengupta, “Lower and Upper Bounds for a Multiple Depot UAV Routing Problem”, *IEEE Conference on Decision and Control*, December 2006

[24] Parker, L. E, “ALLIANCE: An architecture for fault tolerant multi-robot cooperation”, *IEEE Transactions on Robotics and Automation* 14(2), 220–240. 1998

[25] Korte, B. & Vygen, J., *Combinatorial Optimization: Theory and Algorithms*, Springer-Verlag, Berlin 2000

[26] Zlot, R., Stentz, A., Dias, M. B. & Thayer, S. “Multi-Robot Exploration Controlled by a Market Economy”, *International Conference on Robotics and Automation*, Washington, DC, pp.3016–3023. 2002

[27] Chaudhry, R. D'Andrea, and M. Campbell, “RoboFlag - A framework for exploring control, planning, and human interface issues related to coordinating multiple robots in a realtime dynamic environment”, *Intl. Conf. on Robotics and Automation*, 2003

[28] J. Sullivan, S. Waydo and M. Campbell, “Using Stream Functions to Generate Complex Behavior” 2003 *Guidance, Navigation and Control Conference*.

[29] M. G. Earl and R. D'Andrea, “A study in cooperative control: The RoboFlag Drill,"' *Proceedings of the American Control Conference*, Anchorage, Alaska 2002

[30] Squire P.N., Galster, S.M., & Parasuraman, R. “The effects of levels of automation in the human control of multiple robots in the RoboFlag simulation environment.” *Proceedings of the Second Human Performance, Situation Awareness, and Automation Conference*, 2004

[31] Veverka, J. and Campbell, M., “Experimental Study of Information Load on Operators in Semi-Autonomous Systems,” *AIAA Guidance, Navigation and Control Conference*, Austin TX, Aug. 2003.

[32] M. Campbell, F. Bourgault, S. Galster, D. Schneider “Probabilistic Operator-Multiple Robot Modeling Using Bayesian Network Representation” *IEEE International Conference on Robotics and Automation,* April 2007

[33] R. D’Andrea and M. Babish, “The RoboFlag Testbed”, *Proc. of the American Controls Conference*, June, 2003,

[34] D. Schneider, “The RoboFlag Website,” *Cornell University*, October 2003, http://roboflag.mae.cornell.edu/

[35] Campbell, M., D’Andrea, R., Schneider, D., Chaudhry, A., Waydo, S., Sullivan, J., Veverka, J., Klochko, A., “RoboFlag Games using Systems Based, Hierarchical Control,” *American Control Conference*, June 2003.

[36] Kelly, K., Labute P., “The A* Search And Applications To Sequence Alignment”, *Chemical Computing Group Inc.* Montreal, Canada. 1996

[37] Dechter, R., Judea, P., "Generalized best-first search strategies and the optimality of A*". *Journal of the ACM*, 32 (3): pp. 505 – 536, 1985

CHAPTER 3

IMPROVED OPTIMISTIC PREDICTIVE COST METHOD FOR FASTER G*TA REAL-TIME OPTIMAL TASK ALLOCATION

Traditionally, many applications that require solutions to NP-Hard task allocation problems have had to use approximation methods in order to run in real time. Recent advancements however, with the creation of the G^*_{TA} algorithm, have shown that guaranteed optimal solutions can be achieved in average computational runtimes that are up to two orders of magnitude faster than the previous main optimal approach of using traditional MILP implementations. This paper presents a further enhancement to the G^*_{TA}'s optimistic predictive cost function that leads to overall computational runtimes that are on average up to 5 times faster than the original G^*_{TA} method's, hence allowing optimal solutions to be an available option to more real time applications. Both the original method and the new method are explained in this paper in relation to the rest of the G^*_{TA} method to demonstrate their differences and to explain the new method's benefit. Implementation test results are also presented which demonstrate the use of the new optimistic predictive cost method requires up to an order of magnitude less memory on average to achieve its faster, guaranteed optimal results.

3.1 Motivation

The ever increasing need to solve a wide variety of NP-Hard task allocation problems effectively in reasonable computation times has been made clearly evident by the considerable amount of research conducted in this area [3.1]-[3.26]. The range of this research has been as varied as the potential applications, some of the most

prominent being space exploration [3.1],[3.2], coordinated UAV control [3.3],[3.4] and even robotic soccer [3.5]. The approaches to solve these problems can be divided into two basic categories: approximation methods which are relatively fast but may not always produce the most effective solution, and optimal methods which provide a guarantee of finding the "best" possible solution but may take considerably more computation time and memory to execute. One algorithm in the latter category is the G^*_{TA} algorithm [3.7]. This paper extends the number of applications that are able to use optimal methods however by improving the average computation times of the already very fast, guaranteed optimal G^*_{TA} method [3.7] by more than a factor of five through an enhancement to its optimistic predictive cost function.

Traditionally, for many applications where solution speed was a significant concern, an approximation method would be required that might sacrifice accuracy in order to meet the computation time requirements. As this situation is very common, the research area of approximation methods has been growing very quickly, particularly for less complex versions of task allocation problems [3.8],[3.9]. Many notable recent methods make use of a variety of algorithms from other fields including negotiation and even financial models as well and have been implemented by groups at Carnegie Mellon, Stanford, USC and LAAS-CNRS [3.10]-[3.15]. In fact due to its growth, ref. [3.8] provides a summary of many of the other current approximation methods as well as a suggested taxonomy for describing the many problem variations.

In those applications which may require more accurate answers than approximation methods can provide but may not have as strict time requirements, the approach has been to combine optimal methods with approximation heuristics. The main optimal method that is typically used in these approaches has been Mixed Integer

Linear Programming (MILP), as many of the problems are easily translated into the MILP framework. Recent developments using MILP for task allocation problems include work being done at MIT, Wright Patterson AFRL, Berkley and several other leading universities [3.2],[3.16]-[3.23]. Furthermore, due to its wide use, Dr. Sangiovanni-Vincentelli's group at Berkeley has even published a classification of MILP representations of the problem [3.21]. MILP on its own, however, requires significant computation time and memory usage in order to produce its optimal solutions [3.7]. Hence, significant difficulties can occur when they are applied to highly dynamic environments where solutions must be obtained quickly and problems must be completely resolved very frequently.

One example of the combined MILP/approximation method approach is the coordinated reconnaissance work of Ousingsawat & Campbell, which combined MILP with clustering techniques to reduce the problem size [3.24]. Another example is the intercept path planning work done by Earl & D'Andrea which used MILP not to determine optimal assignment, but whether there existed an assignment that met a certain goal [3.22]. Most recently, Rathinam & Sengupta from Berkley have modified Held-Karp's lower bounds TSP method to handle the multiple depot, multiple salesman task allocation problem to a 2-approximation, LP-relaxation method which forces fixed ending tasks [3.25]. Some of the most notable work though has been done at Dr. John How's group at MIT, including a study of MILP experimentations where MILP is used to solve higher level problems at a lower frequency (on the order of seconds) and model predictive control (MPC) is used in between MILP based updates [3.16],[3.23].

Recent work by the author, however, has shown that a new method called G^*_{TA}, that is based on an A^* framework is able to produce optimal solutions in computation times two orders of magnitude faster on average than a traditional MILP implementation. [3.7] As G^*_{TA} is one of the fastest for determining optimal solutions to a variety of NP-Hard task allocation problems, being able to improve this method's average runtime would allow this optimal method to be applied a wider variety of applications that now must rely on combined approaches as mentioned earlier in this paper.

As one of the key features of the G^*_{TA} method is the creation and growth of partial solutions to task allocation problems. These partial solutions, commonly referred to as nodes, are then assessed using an optimistic predictive cost estimate to determine which nodes should be grown further. Hence, any improvement to the runtime of the optimistic predictive cost estimate method or the accuracy of the estimates themselves could help to improve the runtime of the overall G^*_{TA} method.

This paper focuses on the latter approach and contributes a new method for computing the optimistic predictive cost used with G^*_{TA} to produce more accurate estimations. It will be shown in this paper that this new optimistic predictive cost method in turn significantly reduces the exploration required by the algorithm and hence leads to overall G^*_{TA} average computation times that are up to five times faster than the original method while using up to an order of magnitude less memory.

The paper begins in Section 3.2 with a definition of the NP-Hard task allocation problem that is the focus of this paper. Particular attention is given to how this problem definition fits into the problem taxonomy of [3.8] so as to aid in relating this research more directly to other work. In order to demonstrate how the new

optimistic predictive cost function works within the overall G^*_{TA} algorithm, Section 3.3 provides an overview of the principal components of the G^*_{TA} algorithm, as well as a discussion on its guarantee of optimality.

Section 3.4 then describes both the original and new optimistic predictive cost functions. This section is organized into three parts where Section 3.4.1 describes a minimum spanning forest algorithm; used in both the original and new optimistic predictive cost functions. Section 3.4.2 offers a detailed description of the original optimistic predictive cost function so that it can be compared and contrasted with the new method in Section 3.4.3.

For completeness, Section 3.5 describes G_{TA}, the greedy algorithm used in G^*_{TA} to create upper bounds on the final cost. This estimate is used to reduce the search space of G^*_{TA} and was shown in [3.7] to reduce the computation times of G^*_{TA} by up to 70%. Finally in Section 3.6, a series of implementation tests of G^*_{TA} using both the original and new optimistic predictive cost functions are discussed that support the benefits of the new method mentioned earlier.

3.2 Task Allocation Problem Definition

The task allocation problem defined in this paper is the same as the one defined in the previous work [3.7]. In general, "task allocation" is used to describe a variety of problems where there is a given set of tasks (a.k.a. targets, goals, etc) that must be performed, and there is a given set of agents (a.k.a. robots, sources, etc) that can perform the tasks. This paper considers the common form where any task may be assigned to any source and any source may be assigned multiple tasks or none at all.

In the problems considered here, there is a different assignment cost (or utility) for each task/source pair as well as a different assignment cost for transitioning from any one task to another. This last statement connotes that the order in which tasks are assigned to a source does influence the overall cost. In addition, all costs are assumed to be non-negative and the cost associated from transitioning from state A to state B does not have to be the same as the cost associated from transitioning from state B to state A. The problem definition is further generalized to allow it to be more applicable to more real world scenarios by allowing the triangle inequality not to hold. This last point is a generality that is not always commonly handled in task allocation algorithms and will be highlighted later in the discussion.

For completeness, the task allocation algorithms presented in this paper also allow that every task assignment may have an associated list of constraints. These constraints may be either source specific, such as a limit on the amount of a resource that each source can expend, or task specific such as requiring a set resource amount or specific time constraints. These constraints are also allowed to vary depending upon the previous task assignments, i.e. there can be resource expenditures for transitions between tasks. Constraints are also allowed on the problem as a whole, such as requiring that the total resource expenditure across all sources does not exceed a given limit. However this paper only considers problems where there does exist at least one feasible solution.

The topic of constraints in task allocation problems is actually quite involved and therefore this paper focuses itself to problems where the constraints do not influence the solution to the problem but rather the solutions are strictly cost driven.

However as the above mentioned constraints appear in many applications, the algorithms presented in this paper designate how constraint checks are incorporated.

Given the sources, tasks, and the calculated costs and resource / constraint requirements, the goal of the task allocation problem is then to determine which tasks should be assigned to which sources, and equally importantly, in which order those tasks should be assigned, so as to assign all tasks while incurring the overall minimal cost without violating any constraints.

Stated more formally, let s stand for the number of agents (or sources) that can be assigned tasks and let t stand for the number of tasks (or targets) that must be assigned to the sources. Let the means of transitioning a source from its initial state to a target's state and likewise the means for transitioning from one target state to another, be referred to as a "path". Additionally, as discussed further in Section 3.3.1, a path is also assumed to contain all associated assignment and state transitioning costs and constraint information. Finally, let the ordered set of paths assigned to a source be referred to as a "trip", where Tr_i stands for the trip of source i. As a result, all trips always begin with a source and then contain an ordered list of the targets assigned to that source.

Table 3.1: Summary of Key Problem Definition Terms

s	Number of sources
t	Number of targets
path	Means of transitioning for a source from its initial state to a target state or from one target state to another
trip	Ordered set of paths to be determined for each source, where Tr_i stands for the trip of source *i*

In this paper, the symbol "*V*" is used to represent the set of all vertices, "*S*" is the subset of source vertices and "*T*" the subset of target vertices, which contain "*v*", "*s*", and "*t*" members each respectively as shown in the equations below.

$$S \subset V, \quad T \subset V, \quad s.t. \forall V_i \in V \ \ if (V_i \in S) \Rightarrow (v_i \notin T) \tag{3.1}$$

$$v = s + t \tag{3.2}$$

With these terms in place, the task allocation problem can be defined as: Given *s* sources, *t* targets and a set of all available paths, the solution method must create a set of trips for the sources, Tr_i for all $i \in S$, such that each target is assigned to or "visited by" at least one source, without violating any constraints. Furthermore, this assignment must result in the minimal overall combined cost of all of the sources' trips, as expressed in (3.3), where *J* is the overall combined cost and $J_i(Tr_i)$ is the cost incurred from the trip of source *i*. It is also assumed that a "one way trip" is the default problem variation where a source may end its trip at any of the targets or remain at its current state without being assigned any targets at all.

$$J = \sum_{i=1}^{s} (J_i(Tr_i)) \tag{3.3}$$

Defining the problem in this manner allows the problem to be interpreted as a graph problem as shown in Figure 3.1. Using the graph representation the input was standardized as:

1. A highly connected directional graph where each vertex, $v_i = [x_i,y_i]$ is defined as either a source or a target and each graph edge, $e_{ij}(v_i,v_j)$, is assigned a weight, $w_{i,j}$, equal to the cost of the path between vertices v_i and v_j.

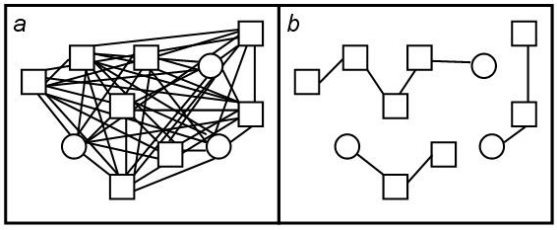

Figure 3.1: Graphical Representation of the Task Allocation Problem with sources as circles and targets as squares a) input as a highly connected graph b) solution as series of selected graph edges to form 3 "trips", Tr_i, $i \in \{1..3\}$

Similarly, the output solution was standardized as:

1. The final cost associated with the optimal solution, J^*.
2. The trip that should be executed for each source, Tr_i, $i \in \{1..s\}$.
3. A report on the constraints and/or resource usage. For this paper's tests, the constraints measured include the amount of fuel used, the overall time it would take for the sources to execute the solution, and the time that each individual task

was completed where again the constraint / resource usage information is provided via the path input information.

4. The solution method's total computation time

In terms of [3.8]'s taxonomy, this described problem is a more difficult version of the Single Task Robots – Single Robot Tasks – Time Extended Assignment or ST-SR-TA case, i.e. single task robots implies that a robot can handle several ordered tasks but only one at a time, and single robot tasks implies that tasks only require a single robot's attention at some point in the task allocation. A fairly well known example of a ST-SR-TA problem is the ALLIANCE Efficiency Problem (AEP) first stated by Parker [3.27]. The problem presented here is a more difficult ST-SR-TA is because of the interdependency between the costs, i.e. whether a robot visits target A or target B first influences the cost to travel next to a target C. With this observation, it becomes apparent that the problem presented here is an instance of the Multiple Depot, Multiple Traveling Salesman Problem and is hence NP-Hard, as was shown by Korte & Vygen in 2000 [3.28]. As the Multiple Depot, Multiple Traveling Salesman Problem description has perhaps the most intuitive meaning, this paper will henceforth refer to any problem or sub-problem fitting this description as an MTSP.

As discussed in the Section 3.1, both optimal and approximation methods [3.1]-[3.26] have been applied to similar variations of this problem including the previous G^*_{TA} work of [3.7]. Therefore, this is a good problem to demonstrate the improvements to the computation time that the new G^*_{TA} optimistic predictive cost function method presented in this paper provides while still guaranteeing optimality.

3.3 The G^*_{TA} Task Allocation Method

In order to understand the role of the new optimistic predictive cost method within the G^*_{TA} algorithm this section provides a summary of the entire G^*_{TA} method [3.7]. Emphasis is placed on the description of both the original and new optimistic predictive cost methods in order to highlight the benefits of the new method.

The G^*_{TA} task allocation method is best described in terms of two separate components as depicted in Figure 3.2 below and explained in detail throughout Sections 3.3 – 3.5. The primary component is built upon the A^* framework and will hence be referred to as A^*_{TA}. Described in Section 3.3.1 – 3.3.4, the A^*_{TA} component functions by growing partial solutions (or nodes) and evaluating these solutions based upon an optimistic predictive estimation cost until an optimal solution is grown and later identified. Section 3.4 details both the original and the new optimistic predictive estimation cost methods. The operation of the A^*_{TA} component is enhanced by the secondary component, a greedy upper bound algorithm, referred to as G_{TA}, which is described in Section 3.5.1. After each iteration of A^*_{TA}, G_{TA} uses the "best" of the nodes created by A^*_{TA}, which is referred to n_{min}, to generate upper bounds on the final optimal solution's cost. Hence, the G_{TA} component is used to reduce the search space of the A^*_{TA} component, and the combination of these two algorithms is described in Section 3.5.2.

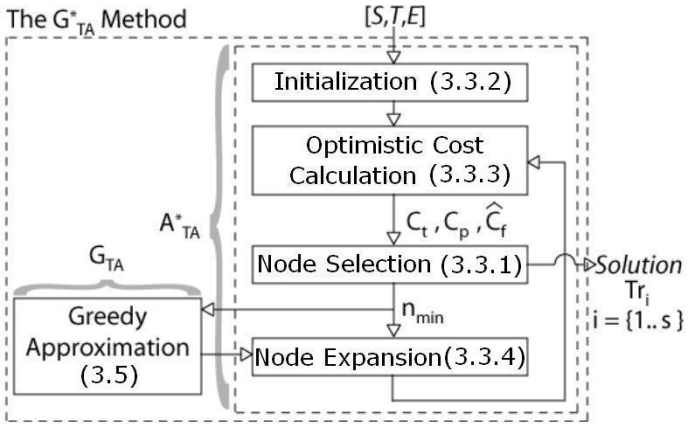

Figure 3.2: The G^*_{TA} Method Separated into it's A^*_{TA} and G_{TA} Components. Details on Each Component are provided in the Section whose Number is Shown in Parenthesis

3.3.1 The A Framework*

The casting of the A^*_{TA} component into the A^* framework can be easily described by first providing a few definitions. A "complete solution" is defined as a set of trips for all sources that together include all of the targets problems, i.e. all targets have been assigned to a source. A "partial solution" is defined as set of trips for all sources that together may or may not include all targets, i.e. there may still be some targets that have not been assigned to a source. From these definitions, a complete solution is considered to be a special case of a partial solution. Furthermore, growing a partial solution is defined as assigning at least one of the partial solution's previously unassigned target to one of the source's trips. Lastly, the final solutions of a partial solution are defined as the set of complete solutions that could be obtained from growing that partial solution.

The most basic function of A^*_{TA} is the growth and evaluation of partial solutions. The most important characteristics of a partial solution are stored as a "node" which is defined in (3.4) as 1) Tr_i, the trip created for source i as it currently stands within the partial solution, 2) $R_{i,j}$, the usage of resource j by source i within the partial solution, 3) r, the number of resource constraints, 4) U, the set of unassigned targets that are not yet part of any trip, and 5) C_t, C_p, and $\hat{C}_f$ which are costs characteristics for the partial solution that will be explained in the A^*_{TA} general overview shown below.

$$node := \begin{Bmatrix} Tr_i \\ R_{i,j} \\ U \\ C_t, C_p, \hat{C}_f \end{Bmatrix} \quad \forall i \in \{1..s\}, \ \forall j \in \{1..r\} \tag{3.4}$$

Following Figure 3.2, the A^*_{TA} components can be summarized as a 5 step process where each step will be explained in further detail in the following sections:

1. Figure 3.2, Initialization: Initialize A^*_{TA} with a new set of nodes from which all possible complete solutions could be grown from. (Section 3.3.3)

2. Figure 3.2, Optimistic Predictive Cost: Calculate the cost to execute just the partial solution. This is known as the node's true cost, C_t. As an example, an arbitrary node is shown in Figure 3.3a where the shown edges represent each source's trip within that node. Similarly, targets with no connecting edges are unassigned targets within this arbitrary node. With this representation, C_t is the sum of all of the shown edges' weights. (Section 3.3.2)

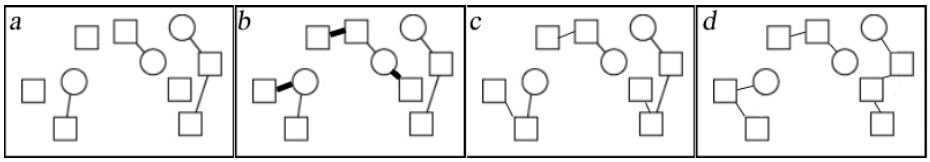

Figure 3.3: Graphical Representation of the Task Allocation Problem using Circles=sources (*s*=3), Squares=targets (*t*=7), and Demonstrating a) an arbitrary node b) optimistic cost prediction for the node in a) as detailed in Section 3.4.2. c) the best possible final solution starting from node a). d) the true optimal solution

3. Figure 3.2, Optimistic Predictive Cost: Create an optimistic predictive estimation of the best remaining cost that would be incurred in completing each new node's partial solution. This is defined as the node's the predictive cost, $C_{p.}$ (Section 3.3.3 and 3.4)

   In the Figure 3.3b example, the thick edges chosen are shown as a way of estimating the best cost to complete the partial solution of Figure 3.3a. The method for choosing these thick edges is detailed in Section 3.4.2, but regardless, C_p in this example is the sum of the thick edges' weights. Please note that this example also shows that the prediction for completing a node does not have to relate to a feasible complete solution but rather only an optimistic complete solution. (Section 3.4)

4. Figure 3.2, Node Selection: Store the newly created nodes in a master set of nodes, N, according to their combined true, C_t, and predictive, C_p, costs as shown in equation (3.5). This combined cost is the final cost estimation, $\hat{C}_f$. (Section 3.3.3)

$$C_t + C_p = \hat{C}_f \leq C_f^* \tag{3.5}$$

   The final cost estimate, $\hat{C}_f$, is also optimistic and therefore it is always less than or equal to the smallest final solution cost possible that could be obtained from starting with that node. This later cost is referred to as C_f^*. The best possible final solution cost starting from the arbitrary node of Figure 3.3a is shown in Figure 3.3c

5. Figure 3.2,Node Selection: choose the node from N with the lowest $\hat{C}_f$; n_{min}
   a. **if** n_{min} is a complete solution, this node is the optimal solution. (Section 3.3.1) The optimal node for the Figure 3.3 example is shown in Figure 3.3d

b. **else** Figure 3.2,Box 4:delete n_{min} from N. Create a new set of nodes that consists of all single step expansions of n_{min}'s partial solution, as demonstrated in Figure 3.4. Return to Step 2 with this new set of nodes. (Section 3.3.4)

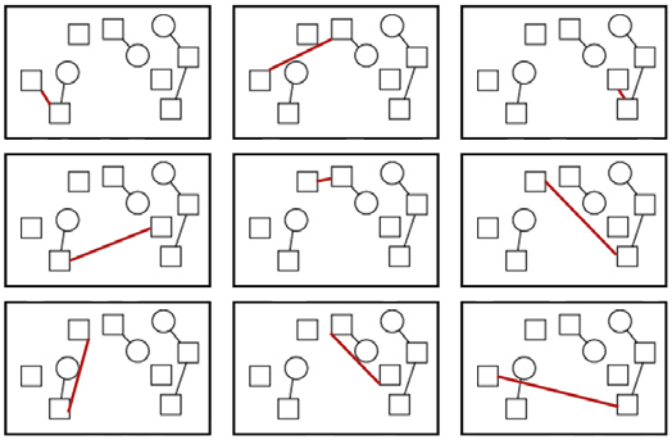

Figure. 3.4: Circles=sources (*s*=3), Squares=targets (*t*=7),
All single step expansions of the arbitrary node of Figure. 3.3a)
The red line in each box is the new single edge
added to the arbitrary node to form a new node

The A^* framework's optimality guarantee has been proven several times (see ref. [3.29] or [3.30] for an example). A short summary of the proof is provided here to support the discussion in further sections. The guarantee that the complete solution from Step 5a. is optimal stems from the A^* framework's optimistic predictive estimation cost requirement. As eluted to in A^*_{TA} component summary, this requirement states that given an arbitrary node, the estimate of the lowest final solution cost for that node, $\hat{C}_f$, must always be less than or equal to the smallest final cost that could be obtained from starting with that partial solution, C_f^*. In the example of the arbitrary node in Figure 3.3a, the estimate of its lowest final solution cost as shown in Figure 3.3b must be less than or equal to what is the actual lowest cost final

solution starting from the arbitrary node in Figure 3.3a. The lowest cost final solution starting from the arbitrary node in Figure 3.3a is shown in Figure 3.3c. Henceforth the Figure 3.3b estimate is called "optimistic". This is also summarized in equation (3.5). Note that the C_f^* for a node is not necessarily the global optimal cost, J^*, as the partial solution specified so far by an arbitrary node is not necessarily a part of the optimal complete solution, as is the case in shown in Figure 3.3a, 3.3c and 3.3d.

For any complete solution, $\hat{C}_f$ equals C_f^* for that node as there is nothing to predict. Furthermore, in the case where n_{min} is a complete solution not only do all other nodes in N have an optimistic $\hat{C}_f$ no better than the $\hat{C}_f$ of n_{min} (by the definition of n_{min}) but by equation (3.5) they must therefore have a best possible complete cost C_f^* no better than the C_f^* of n_{min}. This is summarized in equation (3.6).

$$C_{f,n_{\min},complete}^{*} = \hat{C}_{f,n_{\min},complete} \le \hat{C}_{f,n} \le C_{f,n}^{*} \quad \forall n \in N \tag{3.6}$$

This is obvious for any other complete solutions in N, but it is also true for any other partial solutions. Since all nodes are stored by their estimate cost $\hat{C}_f$, which is optimistic, expanding any other partial solution node at best could only result in a C_f^* equal to its $\hat{C}_f$ which again is already no better than the complete solution cost of n_{min}.

Remembering once again that all possible solutions could be created from the initial set of nodes, (as will be verified in Section 3.3.2), it follows for the ends of (3.6) that when n_{min} is a complete solution, no other node could be expanded into a better solution, and therefore n_{min} is the optimal solution. As will be shown in the sections below, the key to implementing the A^* framework in real-time then becomes being able to calculate C_t and C_p quickly and effectively so that $\hat{C}_f$ is as close as possible to

C_f^*, while remaining optimistic. This concept is revisited in Section 3.3.3 and then again in significant detail in Section 3.4.

3.3.2 Initialization

In the initialization stage of the A^*_{TA} method creates an initial set of nodes to store in N, from which all possible complete solutions could be grown. The simplest set of nodes that could meet this criteria is the set that contains all possible partial solutions which pair exactly one target to exactly one source. For example, consider the simplest case of s=2 and t=2. The top half of Figure 3.5, shows that the initial a set of four nodes for this example that are created by generating every possible assignment of only one target to only one source. This initial set which will always be of the size s x t, is a set of all of the first assignments that could be taken in creating a complete solution. From this initial set, every possible complete solution can be derived, as shown in the bottom half of Figure 3.5.

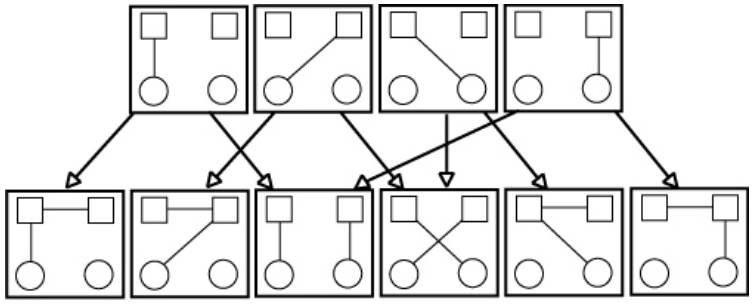

Figure 3.5: Node Initialization and Expansion,
Circles=sources (s=2), Squares=targets (t=2)

As each node is created, the costs C_t and C_p are calculated by means shown in Section 3.3.3 and 3.4, R is updated, and the node is stored in N by its final solution

cost estimate, $\hat{C}_f$. Defining the operations below, the initialization pseudo-code can be written as follows:

n(·)	the **(·)** member of node n
n = MakeNode[n(Tr_i),v_j]	create a new node, n, that extends the end of trip Tr_i, to vertex v_j
CreateUnassigned[v_j,n(U)]	add all target vertices except for vertex v_j to n(U).
CalcCost[n]	calculate C_t, C_p, and $\hat{C}_f$ of node n
Add[n,N]	add node n to the set N such that N will be sorted in ascending order by $\hat{C}_f$

```
1. N = {NULL}
2. for all source v_i ∈ S
3.         for all target v_j ∈ T
4.                 If e_ij(v_i,v_j) exists
5.                         n = MakeNode[n(Tr_i), v_j]
6.                         CreateUnassigned[v_j,n(U)]
7.                         CalcCost[n];
8.                         Add[n,N];
9. return N
```

Pseudo-Code 3.1: G^*_{TA} Initialization

It should be noted that constraints are not checked at the initialization phase because if the edge exists, a path exists between source v_i and target v_j, and the constraints are assumed to have been checked during the path creation prior to the start of the G^*_{TA} algorithm. If the constraints were not checked earlier or if the constraints within the G^*_{TA} algorithm differ significantly from the path creation constraints, line 4 should be modified to include a constraint check.

3.3.3 Calculating Nodes' Final Cost Estimate, $\hat{C}_f$

For each node created, whether it be through the initialization stage or through the node expansion stage of Section 3.3.4, the final cost estimate, $\hat{C}_f$, of that node must be calculated as a sum of the true cost, C_t, and the optimistic predictive cost, C_p, as shown in equation (3.5).

In A^*_{TA}, the true cost, C_t, is calculated as the sum of the $J_i(Tr_i)$ which is the cost of the chosen paths used in the trip of each source $i \in S$.

$$C_t = \sum_{i=1}^{s} (J_i(Tr_i)) \tag{3.7}$$

The second step which calculates the predictive cost, C_p, is more complicated and is described in detail in Section 3.4. Once calculated however, all nodes are stored in the single sorted list, N, according to their final cost estimate, $\hat{C}_f$.

3.3.4 Partial Solution Growth and Node Expansion

After creating an initial set of nodes, the A^*_{TA} component switches to the loop of Figure 3.2 as described below. This loop begins with choosing one node at a time, n_{min}, as described in Section 3.3.1. This section describes the process that then occurs to creates new nodes based on the chosen node. As this process can actually be applied to any node, the subscript "min" on n_{min} will be dropped in this section's discussion. Each new node created from the chosen node, n, is formed by growing one of the chosen node's trips by exactly one of the chosen node's previously unassigned targets, $v_j \in n(U)$. This implies that there is a new node created for every trip / single unassigned target combination and the process of creating a new node for every trip / single unassigned target combination is referred to as a node expansion. An example

of the expansion of the arbitrary node of Figure 3.3a is shown in Figure 3.4. Defining the operations below, the node expansion pseudo-code can be written as follows:

Remove[v_j,n(U)]	remove element v_j from set n(U)
GetBest[N]	return the node,$n \in N$ with the best $\hat{C}_f$; and Remove[n,N]
ExpansionCheck[n,v_i,v_j]	if $e_{kj}(v_k,v_j)$ exists, where v_k is Tr_i's last vertex and the constraints check, return true
RepeatCheck[n,N,X]	examines sets N and the set of all previously expanded nodes, X, for a node with the same final solution cost estimate $\hat{C}_f$ as node n and the same partial solution. If a repeat node is found, return false.

```
1.  K = {NULL}
2.  n = GetBest[N]
3.  while (n(U) != {NULL})
4.          Add[n,X]
5.          for all source vertex v_i ∈ S
6.                  for all target vertex v_j ∈ T
7.                          if ExpansionChecks[n,v_i,v_j]
8.                                  n_new = MakeNode[n(Tr_i),v_j]
9.                                  CalcCost[n];
10.                                 if RepeatCheck[n_new,N,X]
11.                                         Remove[v_j, n_new(U)];
12.                                         Add[n_new,N]
13.                                 else delete(n_new);
14.         n = GetBest[N]
```

Pseudo-Code 3.2: G^*_{TA} Node Expansion

Although fairly straightforward, it is important to note the RepeatCheck function of line 11. As was shown in Figure 3.5, the same partial solution can be

created from expanding different smaller i.e. when the first and last nodes in the top half of Figure 3.5 are expanded they both can create the third node from the left in the bottom half of Figure 3.5. To prevent multiples of the same node from being added to N, the RepeatCheck function checks both the set of unexpanded nodes, N, and the set of all previously expanded nodes, X, to see if the new node, n_{new}, has already been created. It was found in the experimentally in the test of [3.7] that including the RepeatCheck function could improve computation times by up to 75% on average.

The while loop of line 3 demonstrates the exit condition of the overall G^*_{TA} method discussed in Section 3.3.1. According to the A^* framework, if the node that is chosen to be expanded is always the node with the smallest final cost estimate $\hat{C}_f$, the first node to be chosen in line 2 or 14 to have no remaining unassigned targets, i.e. $n(U) = \{NULL\}$, will be the complete optimal solution.

3.4 The Original and New Optimistic Predictive Cost Estimate Methods for G^*_{TA}

With the G^*_{TA} method now discussed, a more detailed description of both the original and the new optimistic predictive cost function methods is detailed. This section first describes the minimum spanning forest algorithm, which is used in both the original and new optimistic predictive cost estimate methods. This is then followed in Section 3.4.2 with a description of the original method of ref. [3.7]. Then in Section 3.4.3, the new method is introduced as a producer of more accurate estimates of the node's final solution. The more accurate estimates, although can only be found via algorithms with larger big "O" runtimes than those of the original method, can lead to faster computation times for the overall G^*_{TA} method.

3.4.1 The Minimum Spanning Forest Algorithm

Both the new and the original optimistic predictive cost methods are based on the minimum spanning forest (MSF) algorithm described here. The MSF is defined as a collection of *s* trees that each minimally span a subset of the entire set of vertices, *V*. The formation of this forest is constrained, however, in that each tree is required to contain exactly one source vertex, and each vertex (source or target) is exclusively a member of exactly one tree. This algorithm is explained first in pseudo-code and then later with the aid of an example in Figure 3.6.

The calculation of the MSF is based on concepts from Kruskal's minimum spanning tree (MST) algorithm [3.31]. Like Kruskal, the MSF algorithm looks to grow and combine trees by first assigning each vertex to its own unique set, with an unique identifier know as the vertex's *setID,* i.e. there is a new set for every vertex and every new set contains only one vertex with its own unique *setID.* This is shown in Figure 3.6a where the numbering and assigning of the setIDs always begin with the sources. Then according to the pseudo-code shown below in , the MSF algorithm follows the greedy nature of Kruskal's algorithm by finding the lowest cost valid edge$\in E$ between any two sets and then merges these sets. Then upon merging sets, all vertices within the newly merged set are given the same *setID* number. This is shown in Figure 3.6b where the two targets of different sets with *setID*s 7 and 5 are merged to one set and both targets now have a *setID* = 5. In the MSF algorithm, the vertices of the new set are always reassigned the *setID* that of the pre-merged sets that is lowest.

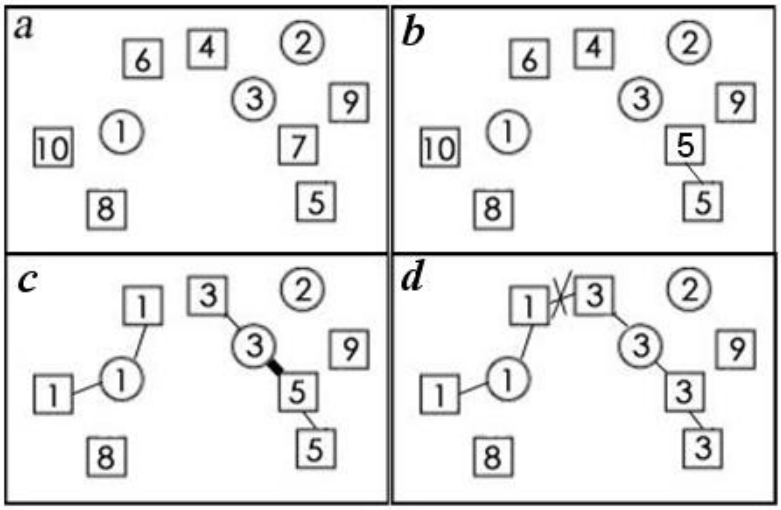

Figure 3.6: a) *setID*s assigned to V as {1..10}, Q={1,2,3} b) merging of target sets with *setID*s of 5 & 7 into one set with *setID*=5 c) during MSF creation, the think line connection between 5 and 3 is allowed as $3 \in Q$, $5 \notin$ Q d)later the connection between elements 1 and 3 is not allowed as both $1 \in Q$ and $3 \in Q$

The rule of always assigning the lowest *setID* as the new set's *setID* is an important that in part defines how the MSF algorithm departs from Kruskal's algorithm. In addition to assigning *setID*s, the MSF algorithm also defines a subset Q as the list of *setID*'s equal to the source vertices'' initial *setID*s. In the MSF algorithm, vertex sets that both have a $setID \in Q$ are not allowed to combine. As the initial setIDs of targets vertices do not belong to the subset Q, target vertices can however combine with each other as well as with the sources' sets. However, if a target set, with a *setID* = j, where $j \notin Q$, combines with a set with *setID* = i, where $i \in Q$, all vertices of the resulting merged set are assigned *setID* = i. This prevents source vertices from ever being within the same set. Hence, as the process of combining sets continues, the number of sets will reduce to $|Q|$, (which also equals $|S|$). At this point, each $setID \in Q$ corresponds to a different MST in the final MSF. The most important part of the

algorithm is to keep track of the edges that are used to create the MSF, as these edges will be used to later calculate the predictive cost, C_p.

MakeSet[v_i]	assigns vertex v_i its own unique set ID.If $v_i \in S$, make the set ID of v_i part of Q
Set[v_i]	returns the set ID of vertex v_i
Union[v_i,v_j]	changes the set ID of all vertices of Set[v_i] to Set[v_j]
Store[e_{ij}, MSF]	make edge e_{ij} part of the set MSF, where MSF $\subset$ E
e_{ij}(v_i,v_j)	The edge, $e_{ij} \in E$, that connects vertex v_i to vertex v_j

```
1.  MSF = {NULL}, Q = {NULL}
2.  for all vertex v_k ∈ V, MakeSet[v_k]
3.  for all edge e_ij(v_i,v_j) ∈ E taken in ascending weight, W_ij, order
4.      if Set[v_i] != Set [v_j]
5           if (Set[v_i] ∈ Q) AND (Set[v_j] ∉ Q)
6.              Union[v_i,v_j]
7.              Store [e_ij, MSF]
8.          else if (Set[v_i] ∉ Q) AND (Set[v_j] ∈ Q)
9.              Union[v_j, v_i]
10.             Store [e_ij, MSF]
11.         else if (Set[v_i] ∉ Q) AND (Set[v_j] ∉ Q)
12.             Union[v_i,v_j]
13.             Store [e_ij, MSF]
14. return MSF
```

Pseudo-Code 3.3: G^*_{TA} MSF Creation

Figure 3.7 shows an example of an initial highly connected graph and the resulting MSF. Again notice that each tree is a MST for the subset of vertices included in that tree, and that there are *s*=3 trees with each tree containing exactly one source.

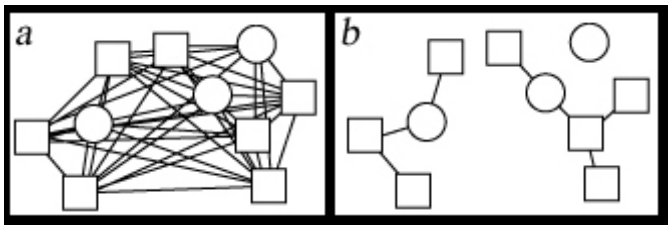

Figure 3.7:a)Graphical representation of a task allocation problem b)MSF resulting from the graph in a) Circles=sources (*s*=3), Squares=targets (*t*=7)

3.4.2 The Original Optimistic Predictive Cost Method

With the MSF subset of edges defined, the optimistic predictive cost, C_p, for any partial solution can now be found. This process is best described with the use of the Figure 3.8 example and then the algorithm is formally defined below. Figure 3.8 shows the same graph and MSF as well as a partial solution that could be created from the same graph in Figure 3.7c. The predictive cost, C_p, for this partial solution is then found by connecting the unassigned targets of the same partial solution to *any* part of that partial solution's trips using the smallest edges of the MSF subset as possible. It follows that the sum of the weights of the used MSF edges then equals $C_{p.}$ This is shown in Figure 3.8d. Although the resulting graphs in Figure 3.8d are not a feasible solution on their own, the important point is that C_p is optimistic because the MSF subset contains the smallest edges possible to connect all of the sources and targets. Hence, there is no better way to assign targets to trips.

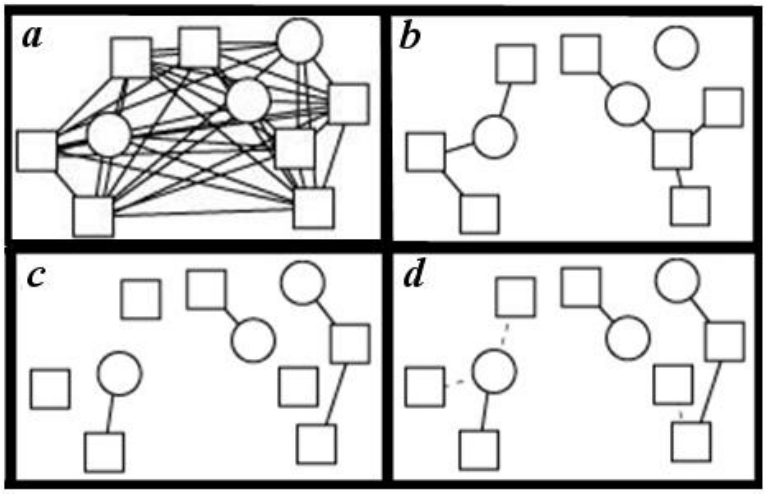

Figure 3.8: a)Graphical representation of a task allocation problem b)MSF resulting from the graph in a) c.)a potential partial solution to the task allocation problem of a) d.)Dashed lines highlight the MSF edges used to calculate the optimistic predictive cost Circles=sources (s=3), Squares=targets (t=7)

Creating the MSF edge subset, assuming that the total edge set E is sorted (as it should be provided from stage 5), is $O(|E|)$ which, from the definition of E in Section 3.2, is nearly equivalent to $O(|V|^2)$. The benefit of creating the MSF subset, however, is that the predictive cost, C_p, of any one partial solution can be calculated using only the $|T|$ MSF subset edges and hence this calculation will be only $O(|T|)$. This new algorithm, which is based off of the original MSF creation algorithm presented in Section 3.4.1, is described in the pseudo-code below. In lines 3-5 all targets that are already part of a source's trip are immediately made a part of that source's set. This is done because the effect of assigning those targets to those sources' trips is already captured in the partial solution's true cost, C_t. Then starting at line 6, those targets not yet assigned to a trip are connected to the existing trips in a tree like fashion using the edges in the MSF subset. Also, since the MSF subset was built with the edges

automatically entered in ascending weight order, no sorting is required in the C_p calculation.

```
1.  Cp = 0
2.  for all vertex vk ∈ V, MakeSet[vk]
3.  for all vertex vi ∈ S
4.      for all vertex vj ∈ Tri,
5.          Union[vi,vj]
6.  for all edge eij(vi,vj) ∈ MSF taken in ascending weight order
7.      if Set[vi] != Set [vj]
8.          if (Set[vi] ∈ Q) AND (Set[vj] ∉ Q)
9.          Union[vi,vj]
10.         Cp += wi,j
11.     else if (Set[vi] ∉ Q) AND (Set[vj] ∈ Q)
12.          Union[vj,vi]
13.         Cp += wi,j
14.     else if (Set[vi] ∉ R) AND (Set[vj] ∉ R)
15.         Union[vi,vj]
16.         Cp += wi,j
17. return Cp
```

Pseudo-Code 3.4: G^*_{TA} Original Optimistic Predictive Cost Function Method

As is discussed in later sections, in cases where there is a relatively high degree of branching in the MSF trees, the effectiveness of the predictive cost, C_p, is lessened, which can increase the A^*_{TA} computation times. The present approach assumes the cost for transitioning between targets is the same for all sources. If the cost is not the same, as may be the case for heterogeneous robots, the algorithm should be modified as discussed in Section 2.8.5.

3.4.3 The New Optimistic Predictive Cost Method

One of the primary purposes behind the optimistic predictive cost method of any A* based algorithms is to provide an accurate estimate of the best final cost if a given partial solution is grown to completion. Hence, the more accurate these estimates, the better the algorithm will perform at deciding which partial solutions to explore and grow further. However, this can be challenging given that the predictive cost method must remain optimistic in order to satisfy the A^* framework's guarantee for optimality. Consequentially, many A^* based methods have to explore significantly large numbers of partial solution nodes in order to find the guaranteed optimal solution. Therefore, traditionally most A^* based algorithms optimistic predictive cost method designs tend to focus more on running very fast, i.e. having a low order big "O" [3.30],[3.32],[3.33].

The new optimistic predictive cost method presented in this paper shows improvement through applying a different approach that it sacrifices the size of its big "O" characterization in return for a more accurate prediction of a given partial solution's best possible final cost. Like the original method, the new method is also based on the concept of a Minimum Spanning Forest (MSF). However, the main algorithmic difference between the two is that unlike the original method, which allowed the unassigned targets of a partial solution to attach to any of the node's sources or targets that were already a part of a trip, as shown in Figure 3.8d, the new method only allows the unassigned targets to attach to the partial solutions trips through the end target or source of each trip. This is not only a valid treatment of the problem, but only growing the MSF trees via the ends of the trips is more closely related to the node expansion process since the G^*_{TA} method will only expand a partial solution's trip through that trip's end. Therefore, since the new optimistic predictive

cost function method more closely resembles the expansion process used to complete a partial solution, this new method is more likely to produce more accurate estimates. The difference between the two methods is exemplified in Figure 3.9, where Figure 3.9a-d mirror Figure 3.8 and the original method, and Figure 3.9e and 3.9f demonstrate the new method's approach.

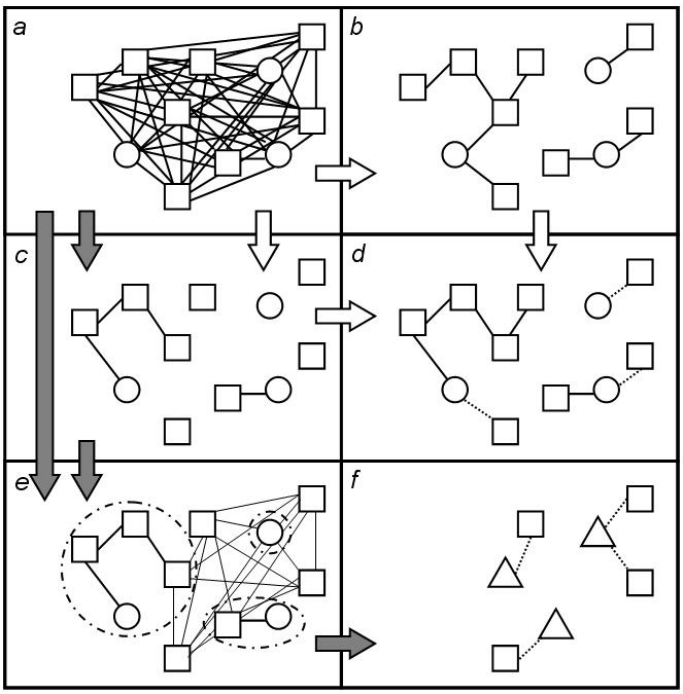

Figure 3.9: Circles=sources (s=3), Squares=targets (t=8), White arrows show original predictive cost method data flow, grey arrows show new predictive cost method data flow. Predictive costs edges used are shown as dashed lines: a)Graphical representation of a task allocation problem b)MSF resulting from the graph in a) c.)a potential partial solution to the task allocation problem of a) d.)Dashed lines highlight the MSF edges used to calculate the optimistic predictive cost e) New optimistic predictive cost graphed edge set, E_{new}, where partial solution trips are surrounded by dash-dot circles to represent their treatment as pseudo-sources. f) Triangles are used to

show how the partial solution's trips are replaced with pseudo-sources in the new optimistic predictive cost method's pseudo-MSF calculation

More explicitly, the new optimistic predictive cost method functions by creating a new MSF for each specific partial solution that only permits the use of edges between unassigned targets and edges extending out of the end of existing trips to the unassigned targets, as is shown in the example of Figure 3.9e. Although this reduced edge set, E_{new}, may seem quite limited as compared to the original input edge set, E, as shown in Figure 3.9a, both sets still allow the use of any unassigned targets edges that could be used to complete the partial solution. Using the reduced edge set, E_{new}, the partial solution can then be considered as an entirely new MSF problem where the new set of input targets, T_{new}, is equal to the unassigned target set for the partial solution, U, i.e. the squares of Figure 3.9f, and the new set of sources, S_{new}, is the end of the partial solution's trips, i.e. the triangles of Figure 3.9f.

In this way, each trip of the partial solution can be reduced to a single pseudo-source with a state which is the same as the last element in that partial solution's trip. The grouping of the trips as pseudo-sources is shown as dashed circles in Figure 3.9e, and the pseudo-source representations of the end of the trips is shown in Figure 3.9f as triangles. This incorporation of the pseudo-sources is a valid representation of the original partial solution, shown in Figure 3.9c, since both the partial solution and the new pseudo-source MSF problem only allow trips to grow from the end of the trips while still preventing any two trips from connecting.

Furthermore an MSF created using the new pseudo-source problem still maintains all of the criteria of the original MSF creation with the exception that the

edge set has been limited to E_{new}, to allow for a more realistic estimate of the final solution from the partial solution. The pseudo-source representation also maintains that no two of the original sources will be connected in the same MSF tree because no two trips are allowed to have a connecting edge in the pseudo-source representation, and each trip (hence each pseudo-source) only contains one source. Hence, an MSF of the new pseudo-source problem, which will be referred to as the pseudo-MSF, represents the best way to connect any of the unassigned targets to each other and to the end of the partial solution's trips using only the edges that could be used to complete that partial solution.

Like the original optimistic predictive cost method, use of the pseudo-MSF may also commonly result in predictions that are impossible to execute. This potential "impossible to execute" nature of the MSF prediction is shown in both Figure 3.9d and 3.9f for the original and new methods respectively, where the source or pseudo-source branches off in more than one direction rather than following an executable (linear) path. However, since by the definition of an MSF, there is no lower cost means of connecting the unassigned targets, the result remains optimistic and the sum of the weight of the edges used in the new pseudo-MSF can be used as the optimistic predictive cost, C_p, for the corresponding partial solution. This calculation of C_p is represented in equation (3.8).

$$C_p = \sum w(E_{newi})) \quad \forall E_{newi} \in pseudo - MSF \tag{3.8}$$

The new pseudo-source optimistic predictive cost method can be created from the modification of the original initial MSF creation algorithm of Section 3.4.1. The inputs to this modified MSF algorithm are the trips as pseudo-sources, S_{new}, the

unassigned targets, U (which is equivalent to T_{new}) and the sorted list of all edges between all sources and targets, E, as defined in Section 3.3, where the limited set of edges, E_{new}, will be obtained via the original edge set, E, according to the methods described below. The pseudo-code for this optimistic predictive cost, pseudo-MSF method is shown below and the differences between the MSF method of Section 3.4.1 and the pseudo-MSF method are then exemplified with the aid of Figure 3.9.

Mark4MSF[v_j]	mark the vertex v_j as being a valid vertex in making a new MSF just for this partial solution's pseudo-sources and unsigned targets
isMarked4MSF[v_j]	returns true if vertex v_j has been marked as a valid vertex for this partial solution by Mark4MSF[v_j]

```
1.  Cp = 0, Q = {NULL}
2.  for all sources si ∈ Snew
3.      MakeSet[si]
4.      if (si is the final vertex of trip Tri)
5.              Mark4MSF[si]
6.      else for all targets tj ∈ Tri,
7.              Union[si,tj]
8.              if (tj is the final vertex of trip Tri)
9.                      Mark4MSF[tj]
10. for all unassigned targets uk ∈ U, i.e. uk ∈ Tnew
11.     MakeSet[uk]
12.     Mark4MSF[uk]
13. for all edge ejk(vj,vk) ∈ E taken in ascending weight, Wjk, order
14.     if ( isMarked4MSF[vj] && isMarked4MSF[vk] )
15.             if Set[vj] != Set [vk]
16.                     if (Set[vj] ∈ Q) AND (Set[vk] ∉ Q)
17.                     Union[vj,vk]
18.                     Cp += Wjk
19.             else if (Set[vj] ∉ Q) AND (Set[vk] ∈ Q)
20.                     Union[vk, vj]
21.                     Cp += Wjk
22.             else if (Set[vj] ∉ Q) AND (Set[vk] ∉ Q)
23.                     Union[vj,vk]
24.                     Cp += Wjk
25. return Cp
```

Pseudo-Code 3.5: G^*_{TA} New Pseudo-MSF Optimistic Predictive Cost Function Method

The method begins by setting set of source *setID*s, Q, to null as was done in the original MSF method and setting the optimistic predictive cost, C_p, to zero as was done in the original optimistic predictive cost method. The new pseudo-MSF method then makes all source's *setID*s members of Q. Then if this source doesn't have any assigned targets yet, i.e. the source's trip ends with the source itself making the source its trip's only member, the source is marked as a valid vertex to be considered later in the algorithm. In this case, the source itself is the pseudo-source.

If the source's trip does contain targets, however, each of those targets *setID* are set equal to that source's *setID* via Union[], in line 7. The target at the end of that trip is also marked as a valid vertex and only edges coming out of that target will be considered as edges for the pseudo-source that is this trip. All unassigned targets are given their own unique *setID*s $\notin Q$ and are also marked as valid vertexes in lines10-12.

All of the vertexes that have been marked valid by Mark4MSF[ ], which consists of the pseudo-sources and the unassigned targets, are then used in lines 13-24 to create a pseudo-MSF of just these vertices. An example of a resulting pseudo-MSF is shown in Figure 3.9f. The drawback to this method is that since every examined partial solution can contain a different set of pseudo-sources and unassigned targets, either a new sorted set of valid edges, E_{new}, must be created each time, i.e. a sorted list of the edges in Figure 3.9e, or the entire sorted edge set, E, must be used. The latter choice of using the entire sorted edge set, E, is used several reasons. First, since the set of pseudo-sources and unassigned targets can be the same as the initial sets of S and T, the big "O" characterization for both edge list options is the same, i.e. $O(|E|\log|E|)$. Hence, the algorithm can e the complete sorted edge list E, which is calculated for other parts of the G^*_{TA} method already. However, as $|S_{new}| + |T_{new}|$ is always less than

$|S| + |T|$ in any G^*_{TA} stage other than the initialization stage, the average time to create new sorted sets of valid edges for only the pseudo-sources, S_{new}, and unassigned targets, T_{new}, will run in less time. In the same way, however, on average only the lower weight edges in E, which are toward the front of the list, will be needed to complete the pseudo-MSF. Therefore both logically, and verified experimentally in Section 3.6, it was decided that using the overall sorted edge list E was more time effective.

Although using E is more effective in the above comparison, it still causes the overall new optimistic predictive cost method to have a runtime of $O(|E|)$ which is significantly greater than the $O(|T|)$ runtime of the original method. However, as Section 3.6 will show, since the new method provides considerably more accurate optimistic predictive cost estimations, significantly fewer nodes will need to be created and investigated. Hence the implementation of the new method will be shown to result in overall faster G^*_{TA} runtimes.

3.5 G_{TA}: The Greedy Upper Bound Component of G^*_{TA}

For completeness, this section provides a description of G_{TA} which as shown in Section 3.3, Figure 3.2, is a greedy based component of G^*_{TA}. This section also demonstrates how the incorporation of G_{TA} allow the creation of upper bound cost estimates throughout G^*_{TA} execution.

3.5.1 G_{TA}: Creating Upper Bound Estimates from a Greedy Algorithm

As mentioned in [3.8],[3.9], greedy approximation algorithms have been widely applied to the task allocation problem for a variety of functions. The greedy algorithm incorporated within the G^*_{TA} method, G_{TA}, is used to try to solve the entire

task allocation problem defined in Section 3.2 very quickly. However, as the full problem is NP-Hard [3.28], G_{TA} obviously cannot guarantee an optimal solution. In fact, because of the presence of constraints, it cannot even guarantee a complete solution. Despite this fact, when complete solutions are generated by G_{TA} , they are used to continually update an upper bound, $\hat{J}$, on the final global optimal cost, J^*. The relationship between $\hat{J}$, and J^*, is shown formally in equation (3.9) where $(Tr_i)^*$ represents the optimal trip for source *i*, and $(Tr_i)_{G_{TA}}$ represents the trip for source *i* as determined by the G_{TA} method.

$$\sum_{i=1}^{s} J\left((Tr_i)^*\right) = J^* \leq \hat{J} = \sum_{i=1}^{s} J\left((Tr_i)_{G_{TA}}\right) \tag{3.9}$$

The pseudo-code for the G_{TA} method by itself is given below.

SortEdges[v_k]	sort all outgoing edges of vertex, v_k
AddTarget[v_j,Tr_i]	add vertex v_j to the end of trip, Tr_i and remove it from U
[v_j,Tr_i]=GetSmallestEdge	returns Tr_i the trip with the smallest edge out of its last vertex to an unassigned target v_j.
ConstraintCheck[v_j,Tr_i]	if the constraints check for adding target v_j to trip Tr_i, return true
Remove[v_j,Tr_i]	remove the edge for the end of Tr_i to vertex v_j from consideration
ValidEdge[Tr]	returns true if there is an edge from the end of any trip, Tr, to an unassigned target.

```
1.  U = { all unassigned targets }
2.  for all vertex v_k ∈ V
3.      SortEdges[v_k]
4.  for all source vertex v_i ∈ S
5.      Tr_i = { v_i }
6.  while (U != {NULL} && ValidEdge[Tr])
7.      [v_j,Tr_i] = GetSmallestEdge
8.      if ( ConstraintCheck[v_j,Tr_i] )
9.          AddTarget[v_j,Tr_i]
10.     else
11.         Remove[v_j,Tr_i]
12. if (U != {NULL} ):  return solution cost as Ĵ
13. else:               return infinity
```

Pseudo-Code 3.6: G_{TA} Greedy Task Allocation Method, the Greedy Upper Bound Component of G^*_{TA}

In lines 2-3, the G_{TA} algorithm begins by creating a separate sorted list of the edges coming out of each vertex. In lines 4-5, the algorithm then initializes its solution with a set of *s* trips, one for each source, where each trip contains only that one source vertex. With these two elements, the G_{TA} algorithm enters into the while loop of lines

6-11, that adds one unassigned target to the end of one of the trips in every complete iteration. This is done in a greedy fashion using the list of edges originating from the end vertex of each trip, $Tr_i(v_{end}(E))$. The lowest weight outgoing edge, $e_{min,i} \in Tr_i(v_{end}(E))$, that connects to an unassigned target is compared for each trip, and the trip with the smallest $e_{min,i,}$ which will be referred to as e_{min}, is expanded to include that edge and the corresponding target. This target however is added *provided* that adding e_{min} and the target does not violate any of that source's trip constraints. If the constraints are violated, the next lowest weight outgoing edge connecting to an unassigned target for this trip is again compared with the other trip's lowest weight outgoing edges, a new e_{min} is determined and the process repeats.

This "while loop" is continued until either 1) all of the targets are assigned to a trip or 2) all of the trips' outgoing edges, $Tr_i(v_{end}(E))$, are investigated and fail the constraint check process. In the first case, a complete solution is found and this solution's cost is returned as the new upper bound, $\hat{J}$. In the second case, no solution is found and a solution cost of infinity is returned.

3.5.2 G^*_{TA}: Combining A^*_{TA} and G_{TA}

The mainstay of combining the G^*_{TA} components is that the G_{TA} component's produced upper bound, $\hat{J}$, can be used to reduce the search space of the A^*_{TA} component. This is done by adding an additional check to the A^*_{TA} RepeatCheck function, described in Section 3.3.4, as follows. If a node's final cost estimate, $\hat{C}_f$ is greater than $\hat{J}$, then this node's partial solution does not need to be investigated any further because the G_{TA} solution has already been found to have a lower cost and still meets the constraints.

This final cost upper bound, $\hat{J}$, can also be updated throughout the execution of the A^*_{TA} component. As A^*_{TA} selects its best node to expand, n_{min}, G_{TA} can be run again using n_{min} as the input. Replacing lines 1-5 of the G_{TA} pseudo-code, as shown in Section 3.5.1, with the code below allows n_{min} to seed G_{TA} with its partial solution.

```
1. U = n_min(U)
2. for all source vertex v_i ∈ S
3.         Tr_i = n_min (Tr_i)
```

Pseudo-Code 3.7: G_{TA} Greedy Task Allocation Modification to Allow G_{TA} to be Seeded with Nodes

With the G_{TA} algorithm seeded, G_{TA} is utilized to complete the solution (although the presence of constraints can occasionally prevent the formation of complete solution from some nodes). If a complete solution can be found and the resulting cost from this run of G_{TA} is less than the current final cost upper bound $\hat{J}$, the current $\hat{J}$ is replaced with this latest G_{TA} solution cost and thereby narrows the search space even further.

It is important to recognize that in the $\hat{J}$ update, the G_{TA} algorithm sort is excluded. This is because the sort has already been done in the first G_{TA} run and therefore can simply be reused. Furthermore, since the sort was done for the edges of each vertex separately, not all of the edges must be reconsidered. Instead, only those at the end of the n_{min}'s trips and for the unassigned targets, $n_{min}(U)$ must be considered. This helps to reduce the computation time of the later G_{TA} calls significantly.

It is noted that using a separate sorted edge list for each vertex actually increases the "big O" running time of the G_{TA} algorithm over using a single sorted list

of all edges. However, for cases where the number of sources, s, and unassigned targets, $|U|$, are small enough, this method actually produces faster results, particularly because the vast majority of the G_{TA} calls are with $|U| < t$ and often $|U| << t$.

If the G_{TA} component never produces a solution, the A^*_{TA} component will still run and produce an optimal answer. However, as shown in [3.7], the extra computational effort spent on running G_{TA} is very effective in reducing the overall average computation time of the G^*_{TA} method.

3.6 Implementation Test Results

The new optimistic predictive cost method of Section 3.4.3 has been shown to provide more accurate estimates of the partial solution's best final cost. However, producing these more accurate estimates was shown to result in a big "O" runtime that is larger than the original method by a factor of $|T|$. Therefore, it is uncertain from the purely theoretical analysis which method will produce faster results. In order to determine the advantages and disadvantages of each method, this section discusses a series of random task allocation tests fitting the problem description of Section 3.2. As a reminder, in all tests, both methods always found identical optimal solutions. Thus, the key comparison in these tests is the computation times.

All tests were performed on a 2.0GHz dual processor PC with 2 GB RAM within the RoboFlag testbed [3.31]. The tests were setup to vary the number of sources $s = \{2..6\}$. Likewise, the number of targets, t, for these tests varied from $t = \{2..9\}$. In general, these are considered data sets of significant size for optimal real-time task allocation [3.7],[3.8]. These are also the ranges of the comparative tests performed in

[3.7] where the original G^*_{TA} was shown to run on average two orders of magnitude faster than a traditional Mixed Integer Linear Programming (MILP) method.

For each possible (s,t) pair, 100 tests were run of non-uniform randomly generated source and target positions over the standard RoboFlag field of 4 meters by 6 meters [3.31]. In general, A^* based methods tend to perform the worst in random environments where there is a lack of structure for the algorithm to exploit. Hence, tests comprised of randomly generated positions will allow this paper to make a conservative presentation of this algorithm's abilities.

The average computation time for selected test sets are displayed in the table and graphs below, where Figure 3.10 and Figure 3.11 are included to illustrate how the computation times scale with respect to $|T|$ and $|S|$ respectively. Table 3.2 also provides data on the average number of nodes involved in finding the solutions.

Table 3.2: Average Computation Time and Node Creation/Explored Comparison of G*TA Using the Original And New Optimistic Predictive Cost Methods.

		Computation Time		Original Method		New method	
s	t	Original	New	Created	Explored	Created	Explored
2	2	0.058	0.057	1.63	1.25	1.38	1.00
2	4	0.269	0.184	12.26	9.75	6.64	3.97
2	6	1.734	0.601	67.56	57.40	19.11	11.44
2	7	5.453	1.243	171.99	156.46	34.37	21.80
2	8	14.358	2.617	399.64	362.77	64.00	41.13
2	9	44.879	5.638	1053.71	961.45	114.23	76.05
4	2	0.087	0.087	2.01	1.20	1.82	1.00
4	4	0.551	0.378	13.70	10.63	7.85	4.26
4	6	6.325	2.144	108.29	96.91	30.91	19.75
4	7	23.348	5.718	337.22	307.12	75.72	44.66
4	8	83.680	14.293	969.54	934.93	131.73	94.32
4	9	278.775	42.886	2776.85	2672.04	310.21	244.22
6	2	0.120	0.119	1.76	1.05	1.70	1.00
6	4	0.823	0.592	14.12	9.99	9.09	4.43
6	6	10.621	3.673	113.85	99.58	37.59	19.34
6	7	39.238	10.097	342.40	319.70	73.36	46.11
6	8	194.601	44.711	1542.66	1488.53	243.45	174.32
6	9	804.696	146.855	5679.91	5571.70	632.46	498.69

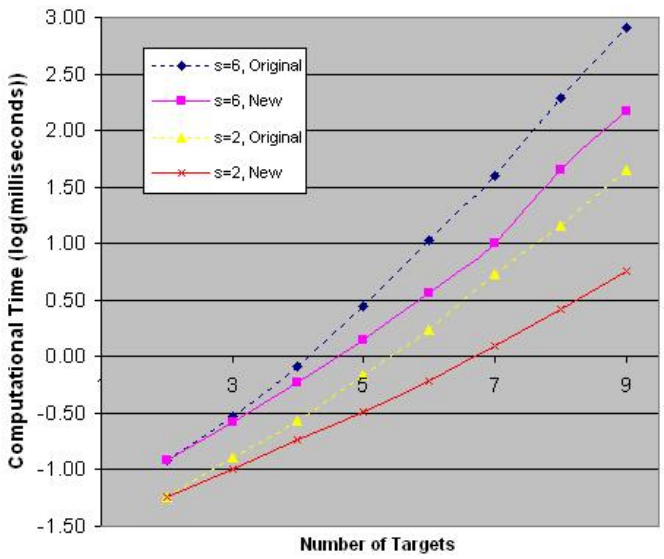

Figure 3.10: Semi-Log Plot of the Average Computation Time vs. The Number of Targets for Various Numbers of Sources for both the Original (dashed lines) and the New (solid lines) Optimistic Predictive Cost Methods in G^*_{TA}

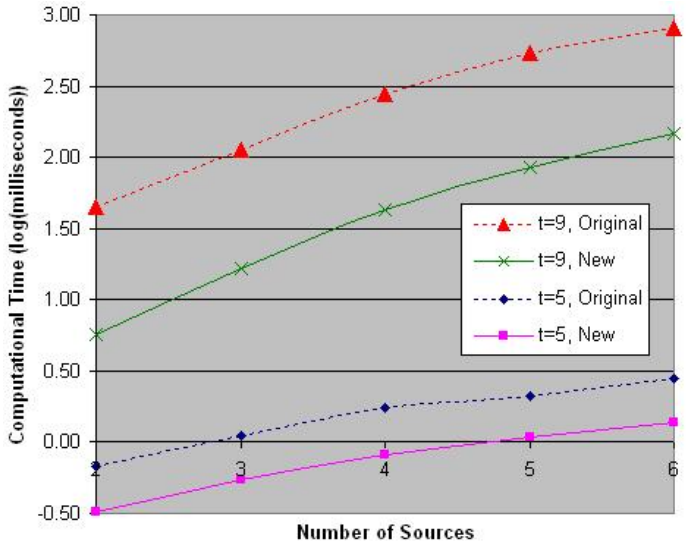

Figure 3.11: Semi-Log Plot of the Average Computation Time vs. The Number of Sources for Various Numbers of Targets for both the Original (dashed lines) and the New (solid lines) Optimistic Predictive Cost Methods in G^*_{TA}

One of the most significant results in this test is that for all (*s,t*) pairs, the new optimistic predictive cost method ran at the same average speed or faster than the original method. More remarkable, however, is that the new method scales significantly better than the original method with respect to increasing either *s* or *t*. This improved scaling is perhaps most clearly shown above in the semi-log graphs of Figures 3.10 and 3.11 by comparing the solid lines (original method) and the dashed lines (new method), where regardless of the number of targets or sources, the new method always ran faster. Furthermore, in the largest test case of s=6, t=9 the new method ran on average more than 5 times faster than the original method.

The scaling benefit can also be seen in examining Figure 3.11 which varies the number of sources for a set number of targets. At the smallest extreme of s=2, t=2, the problems have a far smaller potential solution search space. Hence, as the average number of nodes created by both methods is less than 2, as is shown in Table 3.2, there is little room for improvement by the new method. However, as the number of targets increases, the difference between the two methods varies greatly as can clearly be seen in Figure 3.11. This shift in the two sets of lines shown in Figure 3.11, as well as a comparison of 3.10 with 3.11, also demonstrates that the scaling is more strongly influenced by the size of $|T|$. This is expected however, given the discussion from the earlier sections, particularly the big "O" runtime equations, since $|E|$ grows faster with $|T|$, i.e. since sources connect only to targets, but the targets connect to the sources and the other targets, adding one source add $|T|$ edges but adding one target adds $|S|+|T|$ edges, where $|T|$ is measured before adding the extra target.

The standard deviations for the data presented in Table 3.2 for both methods were at worst slightly less than the average computation time. Although these may seem large, standard deviations of this size are not uncommon in task allocation methods [3.7]. Also, in all cases, the original method's average time was more than $3\sigma_{new\_method}$ above the new method's average computation time.

The average computation time results of Table 3.2 verifies that the extra time spent per node in the new method to produce more accurate optimistic predictive cost estimates leads to faster overall optimal solution times. This finding is supported by analyzing the number of nodes explored and created as also listed in Table 3.2. As the number of targets increases, the number of nodes that must be explored by the overall algorithm in the new method continually decreases as relative to the number of nodes explored by original method. In fact, at t=9 for any number of sources, the number of nodes explored by the new method is smaller by a factor of more than an order of magnitude. This provides strong evidence for why the new method is faster; although the new method requires more time to explore an individual node, the number of explorations required is very small in comparison to the number required by the original method and hence the new method should be faster.

Furthermore, as the number of nodes created is also less, the overall time required to store these nodes in the sorted list, N, as described in Section 3.3.4, is also significantly less. This not only contributes to the faster computation times, but also significantly reduces the storage space.

3.7 Conclusions

The new optimistic predictive cost function presented in this paper for the optimal G^*_{TA} task allocation method has been shown on average to result in computational runtimes up to five times faster than the original method in the studied s={2..6}, t={2..9} experiments. This result has been justified through a discussion on the new optimistic predictive cost function's ability to produce more accurate estimates of the best possible final solution given an arbitrary partial solution and therefore is able to significantly reduce the exploration required by the G^*_{TA} method. These results were validated through a series of implementation tests, which also showed that the new method provides improved scaling of the overall G^*_{TA} method and a reduction in the required memory on average by up to an order of magnitude.

REFERENCES

[1] Campbell, M., "Planning Algorithm for Multiple Satellite Clusters," *Journal of Guidance, Control and Dynamics*, Sept-Oct 2003.

[2] G. Thomas, A. M. Howard, A. B. Williams, and A. Moore-Alston, "Multi-robot task allocation in lunar mission construction scenarios," in *IEEE International Conference on Systems,* Oct 2005

[3] Chandler, P., Pachter, M., *et al.* "Distributed Control for Multiple UAVs with Strongly Coupled Tasks," AIAA *Guidance, Navigation, and Control Conference*, August 2003

[4] Bellingham, J., Tillerson, T., Richards, A., How, J., "Multi-Task Allocation and Path Planning For Cooperating UAVs" *Conference on Coordination, Control and Optimization*, Nov. 2001

[5] Purwin O., D'Andrea R.: "Cornell Big Red 2003", in: Polani D., Bonarini A., Browning B., Yoshida K. (Eds), *Robocup 2003: Robot Soccer World CupVII, Lecture Notes in Artificial Intelligence,* Springer, Berlin, 2003

[6] D. Dyke Weatherington, "DoD UAV Roadmap", *U.S. Department of Defense*, 2003.

[7] D. Schneider, M. Campbell, "Real Time Guaranteed Optimal Task Allocation using Non-MILP Methods" *Paper submitted to IEEE Transactions on Robotics*

[8] Gerkey, B., Mataric, M., "A formal analysis and taxonomy of task allocation in multi-robot systems" *International. Journal of Robotics Research* 23(9):939-954, September 2004

[9] M.B. Dias, R.M. Zlot, N. Kalra, and A. Stenz, "Market-Based Multirobot Coordination: A Survey and Analysis," *Proceedings of the IEEE,* Vol. 94, No. 7, July 2006

[10] P. B. Sujit, A. Sinha, and D. Ghose, "Multi-UAV Task Allocation using Team Theory," in *IEEE International Conference on Decision and Control, and the European Control Conference*, Seville, Spain, December 12-15 2005, pp. 1497–1502.

[11] R. Zlot and A. Stentz, "Complex task allocation for multiple robots," in *International Conference on Robotics and Automation*, Barcelona, Spain, April 2005, pp. 1515–1522.

[12] B. P. Gerkey and M. J. Matari´c, “Sold!: Auction methods for multirobot coordination,” *IEEE Transactions on Robotics and Automation*, vol. 18, no. 5, pp. 758–786, October 2002.

[13] R. Zlot, A. Stentz, M.B. Dias, and S. Thayer, "Multi-robot Exploration Controlled by a Market Economy" *IEEE International Conference on Robotics and Automation*, 2002

[14] T. Lemaire, R. Alami, and S. Lacroix, “A distributed tasks allocation scheme in multi-uav context,” in *IEEE International Conference on Robotics and Automation*, New Orleans, LA, April 2004, pp. 3822–3827.

[15] Zlot, R., Stentz, A., Dias, M. B. & Thayer, S. “Multi-Robot Exploration Controlled by a Market Economy”, *International Conference on Robotics and Automation*, Washington, DC, pp.3016–3023. 2002

[16] Y. Kuwata, A. Richards, T. Schouwenaars, and J.How, "Distributed Robust Receding Horizon Control for Multi-vehicle Guidance" *IEEE Transactions on Control Systems Technology Journal*, accepted 2007

[17] C. Schumacher, P. Chandler, M. Pachter, and L. Pachter, “UAV Task Assignment with Timing Constraints via Mixed-Integer Linear Programming”, *AIAA 3rd "Unmanned Unlimited" Technical Conference, Workshop and Exhibit*, September 2004

[18] M. A. Darrah, W. Niland, and B.M.Stolarik, “Multiple UAV Dynamic Task Allocation Using Mixed Integer Linear Programming in a Sead Mission,” in *Infotech@Aerospace*, Arlington, Virginia, September pp.26-29, 2005.

[19] N. Atay, and B.Bayazit “Mixed-Integer Linear Programming Solution to Multi-Robot Task Allocation Problem” *IEEE International Conference on Robotics and Automation*, 2007

[20] A. Bender. MILP based task mapping for heterogeneous multiprocessor ystem” *In Proceedings of EURO-DAC*, September 1996.

[21] A.Davare, J.Chong, Q. Zhu, D. Densmore, A. Sangiovanni-Vincentelli, “Classification, Customization, and Characterization:Using MILP for Task Allocation and Scheduling” University of California Berkley, Technical Report No. UCB/EECS-2006-166, December 2006.

[22] M. G. Earl and R. D'Andrea, “Iterative MILP Methods for Vehicle Control Problems,” *IEEE Conference on Decision and Control*, Atlantis, Paradise Island, Bahamas, Dec. 2004

[23] Richards, A., Kuwata, Y., How, J., "Experimental Demonstrations of Real Time MILP Control" *AIAA Guidance Navigation and Control Conference*, Aug. 2003.

[24] Ousingsawat, J., and Campbell, M., "Multiple Vehicle Team Tasking for Cooperative Estimation", *American Control Conference*, 2004

[25] S. Rathinam, and R. Sengupta, "Lower and Upper Bounds for a Multiple Depot UAV Routing Problem", *IEEE Conference on Decision and Control*, December 2006

[26] Richards A, Bellingham J, Tillerson M, and How J, "Coordination and Control of Multiple UAVs", *Guidance Navigation and Control Conference*, Aug. 2002.

[27] Parker, L. E, "ALLIANCE: An architecture for fault tolerant multi-robot cooperation", *IEEE Transactions on Robotics and Automation* 14(2), 220–240. 1998

[28] Korte, B. & Vygen, J., *Combinatorial Optimization: Theory and Algorithms*, Springer-Verlag, Berlin 2000

[29] Kelly, K., Labute P., "The A* Search And Applications To Sequence Alignment", *Chemical Computing Group Inc.* Montreal, Canada. 1996

[30] Dechter, R., Judea, P., "Generalized best-first search strategies and the optimality of A*". *Journal of the ACM*, 32 (3): pp. 505 – 536, 1985

[31] Chaudhry, R. D'Andrea, and M. Campbell, "RoboFlag - A framework for exploring control, planning, and human interface issues related to coordinating multiple robots in a realtime dynamic environment", *Intl. Conf. on Robotics and Automation*, 2003

[32] Hart, P. E., Nilsson, N. J., Raphael, B.,"A Formal Basis for the Heuristic Determination of Minimum Cost Paths". *IEEE Transactions on Systems Science and Cybernetics*, SSC4 (2): pp. 100–107, 1968

[33] Hart, P. E., Nilsson, N. J., Raphael, B., "Correction to "A Formal Basis for the Heuristic Determination of Minimum Cost Paths"". SIGART Newsletter 37: pp. 28–29., 1972

CHAPTER 4

CONSTRAINED SIZE, ADAPTIVE, HIERARCHICAL, K-MEANS CLUSTERING FOR THE SUB-PROBLEM DIVISION OF NP HARD PROBLEMS

As the need to solve NP-Hard problems in real-time applications continues to grow, this paper offers a K-means based clustering method that was developed to effectively divide larger NP-Hard problems into a hierarchy of sub-problems where the sum of the sub-problem computation times can be far less than the computation time of the original large problem. Since NP-Hard problems are very sensitive to their input sizes, this new clustering method includes two input parameters that are employed to provide guarantees on the maximum size of each cluster sub-problem as well as the maximum size of the top level of the hierarchy. As improving the computational runtime of NP-Hard problems is the overlying goal of this paper's work, this paper also presents a series of implementation tests that focus on the computational runtime of the new cluster method under conditions that are typical for many NP-Hard problems.

4.1 Motivation

Within the general computer science research community, there has become a greater and greater need to develop algorithms that can solve NP-Hard problems in real time. As often in these cases, improving the runtimes of optimal methods proves to be very challenging [4.1], a wide variety of approximation techniques have become prevalent and popular in numerous applications such as space exploration [4.2],[4.3], coordinated control of UAVs [4.4],[4.5],[4.6] and even robotic soccer [4.7]. In these methods, one approach has been to divide the larger scale NP-Complete or NP-Hard

problems into several separate problems that can be solved in a reasonable amount of time. This can be seen in ref. [4.6] where a task allocation problem is first broken into sub-sets of tasks according to different agent types, and in ref. [4.8] which describes methods that partition tasks based upon the agent's local sensor data.

Many research applications, however, cannot rely on the agent's properties as the means for determining problem sub-sets. Furthermore, for significantly larger NP-Hard problems, it may become necessary to employ hierarchical methods in order to organize the tasks into a series of sub-problems that can each be solved within a reasonable amount of time. The division of the initial large problem into a hierarchy of sub-problems however often requires specific case heuristics. Ref. [4.1] and [4.8] for example provide an excellent survey of many approaches that have been shown to work well for a variety of specific applications and conditions.

In order to provide a more widely applicable, non-application specific means of creating NP-Hard problem data set partitions that relate to sub-problems of reasonable size, this paper offers a new hierarchical clustering algorithm that guarantees the size of every cluster to be within an input bounds and the number of clusters in the top level of the hierarchy to be within a separate input bounds. These input bounds offer the ability to have direct control over the size of the all of the sub-problems that the clusters represent and therefore stronger control over the entire computational time to solve the original problem. Since this algorithm is primarily developed in order to help improve the runtime of more complicated NP-Hard problems, the quality of this algorithm will be largely based on its ability to provide the hierarchical clustering in reasonable computation times while maintaining the guaranteed input bounds.

Numerous existing clustering algorithms were researched by the authors for this study's purposes. Some of the most common hierarchical cluster techniques are based on agglomerative clustering and at first may seem well suited for this paper's goals as they are well known for their speed [4.9]-[4.11]. However, as stated in [4.9], constrained conditions, like the limits the input bounds this paper places on the size of clusters and the size of the top cluster level, can result in the clustering problem to become NP-Complete as well, and approximation methods cannot always guarantee a feasible partitioning of the problem data set.

Another common approach of using fuzzy logic, perhaps the most common being fuzzy c-means, was also investigated. In general however, it is difficult to set the desired specific limits on the cluster sizes using fuzzy methods. This issue has been overcome through a variety of approaches [4.12]-[4.16] however it typically requires additional algorithmic steps to be applied. One such notable method is described in [4.15], where clearly defined clusters are obtained through incorporating Voronoi diagrams with fuzzy c-means. However, as this study is interested in algorithms that can determine the cluster partitioning quickly, the additional time required for these steps prevented these methods from being an effective option.

Other popular techniques suffered from similar issues such as QT clustering, and spectral clustering techniques. The later of these often requires significant matrix operations and Eigenvalue calculations particularly when connection and/or similarity matrixes are of high rank as can commonly occur with NP-Hard problems [4.17]-[4.21]. Likewise, although recent advancements in locality-sensitive hashing and cluster ensembles have been both diverse and been shown to be highly effective in

partitioning a variety of problem spaces, these methods too were considered as having runtimes that were too large given that these methods typically require several initial partitions to work from [4.22]-[4.26].

For these reasons, this paper presents a new method that is a hierarchical, more adaptive version of the well known K-means clustering algorithm. This new method allows for the splitting and merging of clusters, while maintaining that at the end of every iteration, the size of any cluster does not exceed input size limit. The overall algorithm can be broken down into components as shown in Figure 4.1 where the variables shown are defined in Section 4.2. Section 4.2 also provides a more formal definition of the clustering problem of interest as well as formally defining the method's required inputs and other internal parameters. With the problem well defined, the paper describes in Section 4.3 the methods and rules established for initializing these internal parameters from the inputs. Section 4.3 also presents a means for determining the starting centroids for the initial set of clusters. This is then followed by Section 4.4 which describes the point to cluster assignment process.

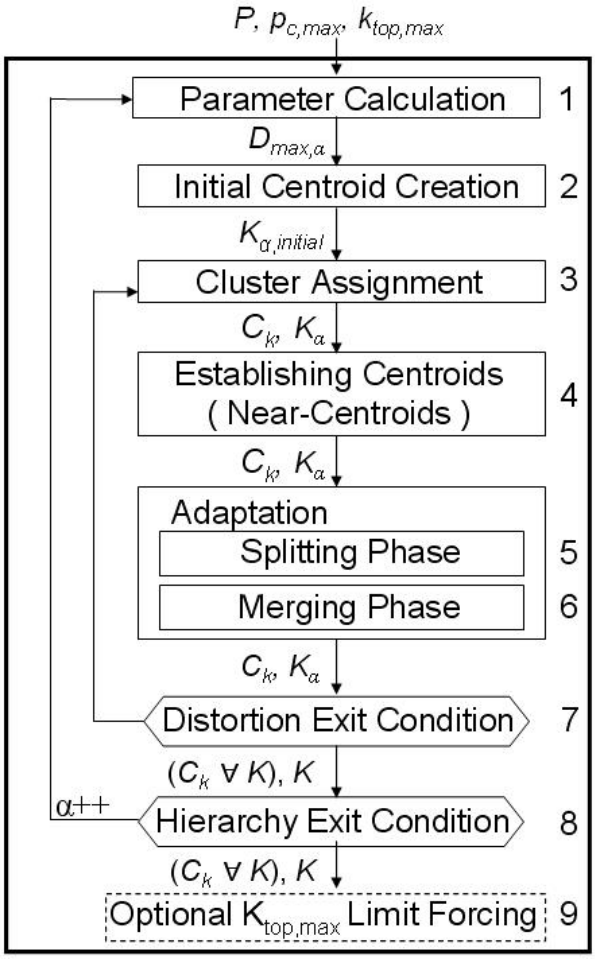

Figure 4.1: Organization of the Algorithmic Components

Once a set of clusters has been created, this set undergoes an adaptation phase that splits clusters that violate the input limits and/or internal parameters, and then merges clusters that can remain within these limits to prevent the hierarchy from growing overly tall. This fairly complex process is designed such that clusters that were just split cannot be immediately re-merged and is described in Section 4.5. Section 4.6 then includes a discussion on the exit condition for the innermost loop of this paper's algorithm as shown in Figure 4.1, with attention also given to the reassignment process within this innermost loop. Section 4.7 then discusses the exit

condition to the overall algorithm, i.e. the outer loop of Figure 4.1, as well as the steps the algorithm takes in adjusting the internal parameters at the start of every iteration of that outer loop. Finally, Section 4.8 describes an optional end step that sets the number of clusters at the top level of the hierarchy to an exact input specified amount. The performance of this new algorithm is then discussed in Section 4.9 through the presentation of a series of implementation tests that focus on larger scale problems that closely resemble a generic setup for many NP-Hard problems, and particularly those found in ref. [4.1] and [4.28].

Throughout this paper, the intuitive 2D problem space, where cost is the distance between points, is used to aid in the explanation of the algorithmic components without loss of generality. For the same reason, specific attention is also given throughout the paper on Traveling Salesman Problems (TSP) or the task allocation problem more formally defined in [4.28] as an MTSP, as these are NP problems that are also well known and the methods used to solve these problem can potentially benefit from this paper's work. Attention is also given to highlight elements of this algorithm that are easily modifiable, to better fit specific applications without sacrificing the performance of other stages of the algorithm, as a means of showing the wide applicability and flexibility of this paper's method.

4.2 Clustering Problem and Input Definitions

Like most clustering algorithms, the input to the method includes the set of points, P, and the edge cost between all points $\in P$. Unique to this method, however, are two additional terms, the maximum points allowed in any one cluster, $p_{c,max}$, and the maximum number of clusters allowed at the top level of the hierarchy, $k_{top,max}$.

It is the later two inputs, $p_{c,max}$ and $k_{top,max}$, that allow the method to control the scale of each sub-problem that the clusters represent. As not only the overall computation time, but the standard deviation of the computation time increases significantly for many complex problems such as TSPs and MTSPs, allowing the programmer to control the scale of the sub-problems provides the programmer with greater control and certainty over the computational run-time. From the four inputs listed above, the rest of the method's parameters, including requirements on the maximum cluster distortion, can be derived within the method, using the equations provided in Section 4.3.

As a requirement to the implementation of this algorithm, it is assumed that there is an external method provided that allows the calculation of the edge costs between all points $\in P$. Likewise, it is assumed for the establishment of the centroids (or near-centroids) as discussed in Section 4.4, that there is an external method for summarizing the points within cluster to determine a potential centroid and for calculating the cost to that summarization point from the cluster's points. Furthermore, for the optional near-centroid method offered in Section 4.4.1, it is also required that there be a method for determining which point within a cluster is most similar to a given summarization point within the problem space. For most cases, however, this last requirement can be satisfied by the requirements for the external cost calculation methods.

Finally, throughout the discussion of this algorithm, let K stand for the set of all clusters and K_α, stand for the set of all clusters at level α.

4.3 Initialization and Calculating Internal Parameters

The number of initial points chosen as potential centroids for the first level of clustering, is determined according to equation (4.1) where, k_0, is the number of centroids at level zero during the initialization phase and is named "k" out of respect for the original K-means algorithm.

$$k_o = P \Big/ p_{c,\max} \tag{4.1}$$

Additional problem information may be used to create an intelligent selection of the location of the first k_0 points. However, for the generic case, a general rule is provided in the Pseudo-Code 4.1 below where P_c stands for the subset of k_0 points that will used later as the initial centroids. This rule first selects either a random point in P or as shown in the Pseudo-Code 4.1, the point in P that is the farthest extreme within the problem's cost space. In the intuitive case of clustering points in a 2D space where the cost is equal to the distance between points, this first selected point could be the one with the largest "x" coordinate.

extremePoint(P)	return the point $\in$ P that is at the farthest extreme in the problem's cost space
makePartofSet(t,P_c)	make point t a part of the set P_c where $P_c \subset P$
farthestFromSetinSet(P_c, P)	returns a point in $P \in P_c$ that is has the largest average cost from all points $\in P_c$

```
1. t = extremePoint(P)
2. makePartofSet(t,P_c)
3. k = 1
4. while(k < k_0)
5.         t_k = farthestFromSetinSet(P_c, P)
6.         makePartofSet(t_k, P_c)
7.         k++
8. k = 0
```

Pseudo-Code 4.1: Initial Clusters' Centroid Creation

The big "O" runtime for this initial centroid selection is $O(|P|k_0 + k_0|P|k_0)$. The $O(|P|k_0)$ term is for determining the initial centroid "extreme point". Likewise the $k_0|P|k_0$ term is for the k_0-1 iterations for determining the rest of the initial centroids, and in every iteration, all points in P must be compared with the up to k_0 members in P_c. Although this rule is somewhat computationally expensive, it does provide a reasonable spread for the initial k_0 points in cost space where there is little additional information. This rule is also used later in the experiments of Section 4.9. If this initialization step is too computationally expensive, randomly selecting k_0 points from P is an acceptable alternative as well, however, it was found experimentally that the use of this rule led to convergence in fewer iterations.

Unlike many standard K-means based algorithms, k_0 in this method is only the starting number of centroids that the first iteration is seeded with. As the cluster splitting and merging routines will show in Section 4.5, the number of centroids for a level may increase, decrease, or remain the same with every iteration.

Although the input variables $p_{c,max}$ and $k_{top,max}$, will be shown to be the main required parameters for determining whether a cluster should be split or merged with another cluster, during the initialization stage an optional parameter that establishes the maximum distortion allowable within a cluster may also be set. In this paper, the distortion of a cluster is defined as the average cost from a point in a cluster to that cluster's centroid and the maximum distortion of a cluster is defined as the maximum cost between any two points in the same cluster. These two definitions are shown below in equations (4.2)-(4.4) where P_k is the set of all points within a single cluster k, the value D_k is the distortion of cluster k, and $D_{max,k}$ is the max distortion of cluster k.

$$D_{k,total} \equiv \sum J_{i,C_k(centroid)} \quad \forall i \in P_k \tag{4.2}$$

$$D_k \equiv {D_{k,total}} \Big/ {|P_k|} \tag{4.3}$$

$$D_{\max,k} \equiv \max\left(J_{i,j}\right) \quad \forall i \in P_k, j \in P_k \tag{4.4}$$

From these definitions, the optional parameter, $D_{max,\alpha}$, which stands for the maximum allowable distortion of any cluster within level α of the cluster hierarchy, is introduced to provide the clustering method with another means for determining a cluster's fitness, and for obtaining the programmer's desired form of clustering. A brief discussion on omitting this parameter is provided in Section 4.9, however, it will be assumed throughout the rest of this paper that $D_{max,\alpha}$ is used in all implementations.

$$D_{\max,\alpha} \equiv \max_{allowable}\left(J_{i,j}\right) \quad \forall i \in P_k, j \in P_k \quad for\ each\ k \in K_\alpha \tag{4.5}$$

This maximum distortion for a given α level, $D_{max,\alpha}$,can be set based upon additional application requirements but (4.6) provides a general rule that can be used in the absence of such requirements. The equation (4.6) rule states that $D_{max,\alpha}$ should

be set as the estimated average min cost required for an arbitrary point, *pt*, to be within *n* points, i.e. the set of the point, *pt*, and the *n* "nearby" points together will all be within a cost of $D_{max,\alpha}$ of each other.

$$D_{\max,\alpha} = \underset{pt \in P}{average}\left(n^{th} \min_{z \in P}\left(J_{pt,z}\right)\right) * \left(\frac{p_{c,\max}}{n+1}\right) \tag{4.6}$$

The derivation of this rule can be explained by considering a problem where the set of all points, *P*, is evenly distributed across the problem's potential space. This is easily conceptualized by examining the rule's implementation within a 2D Cartesian area as shown in Figure 4.2, where the cost is simply set as the distance between points.

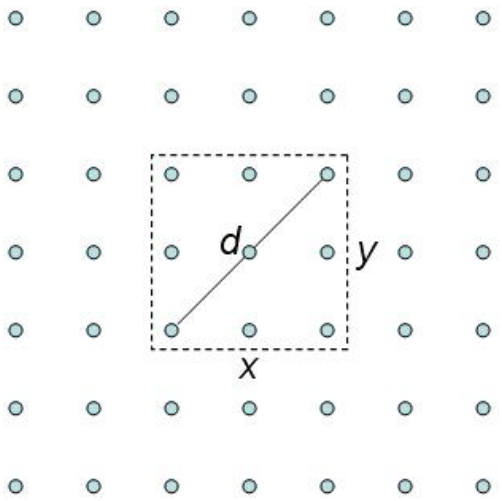

Figure 4.2: Even Distribution of Points in a 2D Cartesian Area Demonstrating D_{max} Determination when $p_{c,max}$ Equals 9

If $p_{c,max}$ is nine, it is very easy to see that clusters will be formed as shown by the dashed line in Figure 4.2. The maximum cost between any two points within this

cluster is also represented in Figure 4.2. as d. Given that the dimensions of the entire space are W and H , x and y can be determined from the fact that the set of points, P is evenly distributed across the space, which results in (4.7) as being an equation for d. In this case, d represents the $\underset{pt \in P}{average}\left(n^{th} \min_{z \in P}\left(J_{pt,z}\right)\right)$ term of (4.6). Therefore, as $p_{c,max}$ for this case is 9 the ratio of $p_{c,max}/(n+1)$ is 1, and $D_{max,\alpha}$ equals the $\underset{pt \in P}{average}\left(n^{th} \min_{z \in P}\left(J_{pt,z}\right)\right)$ which is again just d.

$$d = 2\sqrt{\frac{2HW}{|P|}} \tag{4.7}$$

If $p_{c,max}$ is five, however, it is more difficult to calculate $D_{max,\alpha}$ directly from this problem cost space. Therefore to calculate $D_{max,\alpha}$ according to the rule in equation (4.6), $D_{max,\alpha}$ is first calculated according to equation (4.7) as d, for the $p_{c,max}$ = 9 point case. Then the $D_{max,\alpha}$, for the $p_{c,max}$=5 case is obtained according to the $p_{c,max}$ ratio for the two cases, i.e. $^{5}/_{9}$. It is important to note, in the equation (4.6) rule it is assumed that $1 < n < |P|$ and that this rule works best for $n << |P|$.

Although this description has been specific to this 2D example, it demonstrates the implementation of this rule, without loss of generality, to any problem where the points can be evenly distributed across the problem space. Furthermore, due to the 2D problem realm's intuitive nature, the 2D problem space will be referred to throughout the paper and its figures for descriptive purposes.

4.4 Cluster Assignment

Cluster assignment is handled through a means very much in line with traditional K-means methods. Once the initial set of starting centroids has been established, all points are assigned to the centroid that they have the lowest cost edge to, without any regard to $p_{c,max}$ or $D_{max,\alpha}$. The parameters, $p_{c,max}$ or $D_{max,\alpha}$ are ignored at this stage, to allow all points to be assigned to their own best possible centroid without interference. It is only after this unrestrained assignment occurs that $p_{c,max}$ and $D_{max,\alpha}$ are considered in the cluster adaptation phases of the clustering algorithm of this paper.

4.4.1 Establishing Centroids or Optionally Near-Centroids

Once all points have been assigned to a cluster, the cluster's centroids are adjusted to better represent themselves as a summary point for all points within the cluster. Referring again to the intuitive 2D problem space discussed above, a cluster's centroid would simply be at the average x and y position of all of that's cluster's points.

In many problem spaces, a pure average of the cluster's points may be a suitable means for meeting the Section 4.2 problem definition requirement of establishing the centroid as a summarization of its cluster's points. However, in many cluster applications, attempting to summarize a cluster's points' states may result in an infeasible state within that problem space. This can be seen with a slight adjustment to the intuitive 2D example problem space as shown in Figure 4.3. In this modified 2D space, the points are considered to be drop-off locations for a delivery truck. The average of the clustered locations, in this case, places the centroid within a rock

unreachable by the truck. Hence, the cost to reach the centroid from outside, or rather to go between the cluster's points and the centroid, becomes an impossible value.

Situations like these, can be handled on a case by case basis by adapting this paper's algorithms, but this paper offers a generic additional step that can be applied to any problem meeting the problem definition requirements of Section 4.2. In this additional step, the algorithm determines by the use of the required external methods listed in Section 4.2, the point within the cluster that is most similar to the ideal centroid. As this closest point may not actually be at the same state as the ideal centroid, this closest point is referred to as the near-centroid. Selecting a near-centroid guarantees that the point is indeed a feasible state within the problem space, otherwise the original point would not have existed to begin with. The near-centroid of modified 2D problem space of Figure 4.3 is shown as a triangle in Figure 4.3b.

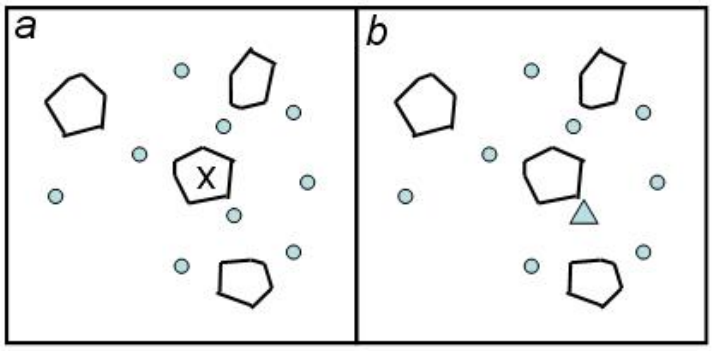

Figure 4.3: A clustered set of delivery points where the polygons are non-traversable rocks a) Situation where the centroid is in a rock and therefore at an infeasible state b) Corrected situation using a near-centroid, shown as a triangle

In general, near-centroids offer the best approximation of the ideal centroids in applications where $p_{c,max}$ is large and the points are well distributed within the cluster

so that the near-centroid and the ideal centroid are very similar to one another. However, in situations where the clusters are fairly sparse and the near-centroids may significantly differ from the ideal centroid, the method can still be used but there is a higher likelihood for the need of adding a special exit condition to the inner loop of Figure 4.1 which will be discussed in Section 4.6.

4.5 Cluster Adaptation

Once all of the points have been assigned to centroids, these newly formed clusters are examined in two adaptation phases, splitting and merging. These adaptation phases not only look to improve on the overall clusters' distortion but they also ensure that the algorithm's cluster size requirements are guaranteed at the end of every assignment/adaptation iteration; no cluster will contain more than $p_{c,max}$ points and no cluster will contain two points whose cost between them exceeds $D_{max,\alpha}$.

4.5.1 Cluster Splitting

The first adaptation phase is cluster splitting. This section reviews every cluster established in the assignment phase to see if that cluster violates either the $p_{c,max}$ or $D_{max,\alpha}$ requirements. Any cluster that is in violation is split, such that the two points within the cluster that have the greatest cost between them are set as new centroids for each new resulting cluster. The points that were a part of the original cluster are then assigned to one of the two new centroids without regards to $p_{c,max}$ or $D_{max,\alpha}$.

After all clusters inputted to the splitting phase have been investigated, all newly formed centroids within the splitting phase are also reviewed for $p_{c,max}$ and/or $D_{max,\alpha}$ violations within the same iteration. If required, these newly formed centroids

are split once. It is only once that all clusters have $p_{c,max}$ or less points within them and no two points within a cluster have a cost greater than $D_{max,\alpha}$ between them, that the set of clusters are allowed to exit the splitting phase. This is represented more formally in Pseudo-Code 4.2 below.

split(P_k)	returns the two members of the set of points P_k that have the greatest cost between them as centroids to new clusters
add2EndOfSet(C_{new1}, α)	adds the cluster **C_{new1}** to the end of the cluster set α
assign2Clusters(P_k,{C_{new1},C_{new2}})	assigns the set of point P_k to the set of the two clusters, **C_{new1}** and **C_{new2}**, based upon the cost to the centroids only
determineCentroid(C_k)	set the centroid (or near-centroid) for the input cluster C_k

```
1. for all k ∈ α
2.       if ( (|P_k | > p_c,max) || (D_k > D_max,α) )
3.                C_new1, C_new2 = split(P_k)
4.                add2EndOfSet(C_new1, α)
5.                add2EndOfSet(C_new1, α)
6.                assign2Clusters(P_k, { C_new1, C_new1})
7.                determineCentroid(C_new1)
8.                determineCentroid(C_new2)
```

Pseudo-Code 4.2: Clusters Adaptation Splitting Phase

As it is possible in the worst case that all clusters could be split perfectly in half in every iteration with the final set of clusters all have exactly one point, the runtime for this method is O(|*P*|log|*P*|). However, this worst case is only realized in cases where the input parameters were poorly set. In the implementation test results

shown in Section 4.9, actual average runtimes of the splitting phase more closely approximated an upper bounds of $O(2|P| / p_{c,max})$.

4.5.2 Cluster Merging

The second adaptation phase is the merging phase. In most cases, the splitting phase tends to increase the number of clusters overall. In general, however, it is desirable to have fewer but fuller clusters, (i.e. clusters with as close to $p_{c,max}$ points without violating any other requirements) so as to reduce the point set representation size faster and so that the overall hierarchical clustering method may require fewer levels. In order to aid in achieving this goal, the merge phase examines the resulting distortion that would occur from merging every pair of clusters that would not violate $p_{c,max}$ once merged. This is performed through lines 2-9 of Pseudo-Code 4.3 below. Then, in Pseudo-Code 4.3 lines 10-17, starting with the cluster pair that results in the smallest distortion increase without violating $D_{max,\alpha}$, the merging phase merges the two smaller clusters into one cluster.

For every iteration of Pseudo-Code 4.3, each cluster is only allowed to be merged once. This is an important point as a single cluster may be part of several cluster pairs that could potentially be merged. That single cluster, however, is only prevented from merging more than through marking each cluster as having been part of a merging in Pseudo-Code 4.3 lines15 and 16, and then looking for that marking in Pseudo-Code 4.3 line 12. If any merging did occur throughout Pseudo-Code 4.3 lines11-17, all clusters at the end of that block are sent back to Pseudo-Code 4.3 lines 1-9 to be investigated again. If any mergable cluster pairs are identified, the marking of the cluster members of those pairs is reset in Pseudo-Code 4.3 lines 8-9.

maxDistortion(C_k,C_m)	return the maximum distortion between any two members within the combined point set of the two clusters' C_k and C_m
add2SortedSet({C_k,C_m},D_{check},K_{pair})	add the paired cluster points C_k and C_m to the cluster pair set $\mathbf{K_{pair}}$ sorted according to each pair's D_{check} value
merged(C_k)	returns true if the cluster point set C_k has already been merged with another cluster point set
mergeClusters(C_k,C_m)	returns the new cluster that contains all the points of C_k and C_m

```
1.  M = {NULL}
2.  for all k ∈ α
3.      for all m ∈ α, m!= k
4.          if ( |Ck| + |Cm| <= pc,max)
5.              Dcheck = maxDistortion(Ck,Cm)
6.              if (Dcheck <= Dmax,α)
7.                  add2SortedSet({Ck,Cm}, Dcheck, Kpair)
8.                  merged(Ck)  = false
9.                  merged(Cm) = false
10. if (M != {NULL})
11.     for all {Ck,Cm} ∈ Kpair
12.         if ( !merged(Ck) && !merged(Cm) )
13.             Cnew1 = mergeClusters(Ck,Cm)
14.             determineCentroid(Cnew1)
15.             merged(Ck)  = true
16.             merged(Cm) = true
17.             add2EndOfSet(Cnew1, α)
18.     Goto Line 1
```

Pseudo-Code 4.3: Clusters Adaptation Merging Phase

The merging phase ends when there are no two cluster pairs that can be merged without violating $p_{c,max}$ and/or $D_{max,\alpha}$. Hence, all clusters are guaranteed to be within the $p_{c,max}$ or $D_{max,\alpha}$ requirements upon exiting the merging phase. This is shown more formally via the first two if statements in Pseudo-Code 4.3, which establish the conditions of merging so that no cluster that is formed from merging would ever be split if run through the splitting phase.

In the very worst runtime case, the input to this stage would be all points $\in P$ as their own separate cluster, and all clusters would be mergable eventually into one single large cluster. Assuming the worst merging sequence as well, this would lead to a runtime of $O(|P|^2\log(|P|^2))$ but these worst cases would only occur if the overall method's inputs were very poorly chosen. In the experimental tests presented in Section 4.9 merging occurred at approximately one quarter the rate that splitting did, i.e. at an approximate upper bound of $O(|P|/(2\ p_{c,max}))$, and hence was not overly runtime intensive.

4.6 Re-Assignment and Exit Conditions

At the end of the splitting and merging adaptation phases, the distortion of each cluster is determined by equation (4.2). This equation states that the distortion of a cluster is defined as the sum of costs from each point in the cluster to that cluster's centroid (or near centroid if that option is being implemented). The distortion of all clusters for an arbitrary level, α, are then summed and stored in $D_{total,\alpha}$, as is shown in equation (4.8). $D_{total,\alpha}$ is the only value that is necessary to calculate in order to examine the exit conditions since, as was stated in Section 4.5, the $p_{c,max}$ and $D_{max,\alpha}$ requirements have already been met.

$$D_{total,\alpha} \equiv \sum D_k \quad \forall k \in K_\alpha \tag{4.8}$$

4.6.1 Exit Conditions

There are two exit condition options for the algorithm that can be used either together or individually. The first and easiest is a traditional threshold condition. If the value of $D_{total,\alpha}$ is below a given $D_{total,max}$ the clustering for this level of the algorithm ends and the next level of the cluster hierarchy is determined according to Section 4.7. This exit condition however should only be employed within problems where it is certain that the threshold can be met or can be met within the computation time requirements of the implementation.

The second exit condition is more general, and can be used when specific distortion threshold information is not available. This second condition examines the delta of $D_{total,\alpha}$ for every iteration of the given level of clustering (Figure 4.1, Blocks 3-7). If the change in $D_{total,\alpha}$ from the last iteration to the most recent one is below a delta threshold, $D_{total,\Delta}$, this level of the clustering ends and the algorithm continues onto the overall hierarchy exit condition of Figure 4.1, Block 8. Although more generally reliable for a wider variety of applications, there is a potential drawback to using this condition that will be explained and become more apparent after the discussion on point reassignment in Section 4.6.3.

4.6.2 Reassignment

If the exit condition(s) fail, the algorithm returns to assignment phase, using the centroids (or optionally near-centroids) as the clusters that all points can be assigned too. Just as in the initial assignment phase, reassignment to clusters occurs without any regard to $p_{c,max}$ and $D_{max,\alpha}$. Once each point has been assigned to the

centroid (or near-centroid) that each point has the lowest cost to, the reassignment phase ends and the adaptation phase is reentered.

4.6.3 Practical Implementation of the $D_{total,\Delta}$ Exit Condition

Due to the reassignment phase being independent of $p_{c,max}$ and $D_{max,\alpha}$ and then always being followed by the splitting and merging phases, the drawback of the exit conditions is that there lies the possibility of incurring cycles of cluster configurations. This is particularly possible with the implementation of the near centroid option, since the near-centroid typically shifts the cluster's centroid from its ideal summary state and this shift can prevent the settling that is normally seen in K-means styled clustering algorithms. The potential of these cycles therefore requires that a method be implemented to store and examine cluster configurations for repeats which would indicate a cycle. If a cycle is identified, this level of the cluster hierarchy is then allowed to exit with a stored cluster configuration of minimum distortion.

Employing hash tables or trees sorted by $D_{total,\alpha}$ are examples of simple but effective methods for storing and sorting cluster configurations. However, if the costs between points are substantially unique it is unlikely that any two cluster configurations will result in exactly the same $D_{total,\alpha}$. Therefore, a practical, more space efficient, but not guaranteed solution to preventing cycles is to keep track of the minimum total distortion at the end of every Figure 4.1 Block 3-7 iteration. Then, if that same minimum is seen again it is reasonable to accept the clustering configuration that produces this observed minimum as the exit configuration.

If this practical loop check criteria is implemented the authors also recommend that an iteration limit be placed on Figure 4.1 Block 3-7, such that if the number of

iterations exceeds that limit, Figure 4.1 Block 3-7 will simply exit with the cluster configuration having the lowest found total distortion. This iteration limit was set in all of this paper's experiments to be 25 iterations, but in the ten thousand trials presented in Section 4.9, the iteration limit exit criteria was never once executed, hence experimentally validating the above practical cycle-handling exit condition.

4.7 Hierarchical Level Iteration

4.7.1 Overall Exit Condition

Once the method exits from creating the clusters for any given level, the overall method exit condition is queried. This condition states that the number of clusters at the top (current) level must be less than or equal to $k_{top,max}$ as defined in the problem definition of Section 4.2 and stated in equation (4.9) where K_{top} is the set of clusters at the top level of the hierarchy.

$$| K_{top} | \leq k_{top,\max} \tag{4.9}$$

If the exit condition is met, the algorithm passes on to the optional $k_{top,max}$ limit forcing phase. If the exit condition is not met, the algorithm enters the super-clustering initialization phase.

4.7.2 Super-Clustering Initialization

In creating the next level of the cluster hierarchy, all of the centroids (or near-centroids) for the clusters of the current level will be treated as the set of points to be clustered in the upcoming level. In the upcoming level, both input parameters $p_{c,max}$ and $k_{top,max}$ will remain the same. However, the maximum distortion between points for the upcoming level will be represented as $D_{max,\alpha}$+1.

4.7.3 Establishing Costs Between Clusters

The method for establishing costs between clusters can also be application specific, but the most general form presented in this paper is the cost between centroids (or near-centroids) which is readily obtainable from the problem definition requirements of Section 4.2.

Extending the intuitive 2D case to this phase, the cost would simply be calculated as the distance between centroids; or in the near-centroid case, this cost is already calculated as the costs between the actual points $\in P$ at the near-centroid. An interesting effect of using near-centroids is that the centroids of all clusters at the very top correspond perfectly to points in the original "bottom" set of points, *P*. This can be a very useful additional outcome provided by this clustering method as these top level near-centroids have now been identified as key representational points for the entire set *P*.

4.7.4 Adjusting D_{max} Per Level

As each cluster at an arbitrary level, α, contains its own centroid (or near-centroid), even the minimum cost between centroids may very well exceed $D_{max,\alpha}$. Therefore, in order to allow the next level of clusters to contain several centroids, the maximum distortion allowable within any cluster of the upcoming level, represented as $D_{max,\alpha}+1$, must be adjusted as well.

This is the basis for the rule of equation (4.10), which is explained with the aid of Figure 4.4. This rule increases $D_{max,\alpha}$ for the upcoming level such that $D_{max,\alpha}+1$ will be the minimum cost between centroids that will allow two clusters of level α to be

included within the cluster for the next level, with one cluster at the current level α between them as the new centroid for the next level cluster. $D_{max,\alpha}$ +1 is always calculated in terms of the original D_{max} for the very first level of the hierarchy, where this original D_{max} is represented in equation (4.10) as $D_{max,0}$ and $D_{max,0}$ can be calculated using equation (4.6).

$$D_{\max,\alpha+1} = 2\left(3^{\alpha}\right)D_{\max,0} \tag{4.10}$$

The relationship between $D_{max,\alpha}$ +1 and $D_{max,0}$ is shown in Figure 4.4b, for moving from the first level of clustering to the second, and in Figure 4.4c for moving from the second level to the third. It is then easier to recognize the pattern established by equation (4.10) where it is assumed that the first level of clusters is referred to as α = zero.

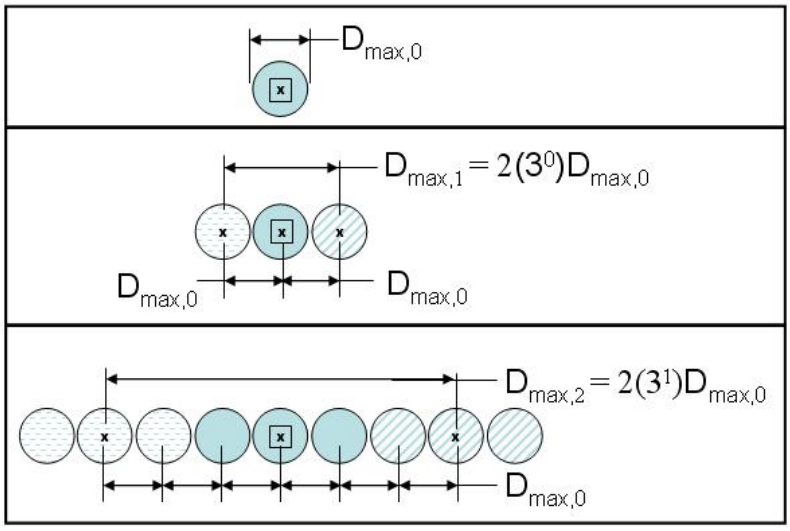

Figure 4.4: Increasing $D_{max,\alpha}$ for each level of the hierarchy as a function of $D_{max,0}$. Clusters at the bottom level are represented as circles, clusters at higher levels are grouped according to their shading. Clusters centroids at the current level are represented as an "x". The centroid for the next level cluster is represented as a box.

The rule of equation (4.10) is written here with the general assumption offered in the previous section, Section 4.7.3, that the cost between clusters is the cost between centroid states. Although other cluster to cluster cost substitution methods can be made, the concept behind the rule of equation (4.10) can still be applied, but the exact equation may need to be modified for these particular cases.

4.8 Optional $k_{top,max}$ Limit Forcing

The hierarchical clustering exit condition listed in Section 4.7 in equation (4.9) states that the algorithm cannot exit until the number of top level clusters is less than or equal to $k_{top,max}$. As an optional post-phase however, the algorithm can force the top layer of clusters that met this exit condition to have exactly $k_{top,max}$ clusters, assuming that $|P| \geq k_{top,max}$.

In general, this optional post-phase makes use of the splitting phase algorithm of Section 4.5.1 to split the top layers clusters so that there are exactly $k_{top,max}$ clusters. It does this by first finding the number of clusters that need to be added, which will be referred to as $k_{top,add}$, which is equal to $k_{top,max}$ minus the number of top clusters as shown in equation (4.11).

$$k_{top,add} = k_{top,\max} - |K_{top}| \tag{4.11}$$

If $k_{top,add}$ is greater than zero, all of the current top clusters are investigated to find the $k_{top,add}^{\text{th}}$ cluster member pairs with the largest distortion between them. The $k_{top,add}^{\text{th}}$ worst pair distortion is then set as $D_{max,\alpha}$ for this top level and then the splitting adaptation phase only is run. As $D_{max,\alpha}$ has just been set to $k_{top,add}^{\text{th}}$ worst edge between

members, the splitting phase will cause up to $k_{top,add}$ splits in the top clusters resulting in up to $k_{top,add}$ new clusters for a total that is always less than or equal to $k_{top,max}$ top level clusters, assuming that all cluster member pair's distortions are unique.

In most cases, a single iteration of this post-phase is sufficient, as was the case for over 99% of the tests conducted for Section 4.9. However, it is possible that if several of the worst distortion cluster member pairs are within the same cluster that this post-phase will need to be run again, with a newly calculated value of $k_{top,add}$. In the worst scenario, this post-phase will only have to run $k_{top,add}$ times, where the value of $k_{top,add}$ is measured prior to the first iteration of this post-phase. If this case is anticipated, or if the cluster member pair's distortions are not unique, it is recommended that only the cluster with the largest cluster member pair's distortion be split and that this procedure is repeated $k_{top,add}$ times instead.

As is true with almost all hierarchical methods, the more levels to the hierarchy and in general, the smaller each level, the more the output of the hierarchical method is an approximation of the points it represents, rather than an accurate summary. Therefore the advantage of implementing this forced top level size is to provide as much detail as possible at that top level to within what the programmer has deemed to be an acceptable complexity.

4.9 Implementation Tests and Results:

This algorithm's development was inspired by the need to help improve the runtime of more complicated problems, like MTSPs, such that this algorithm can intelligently divide a large complicated problem into a far more manageable hierarchy of sub-problem clusters. As described in the previous sections, this algorithm

guarantees that the input properties that the maximum number of points in any cluster will not exceed $p_{c,max}$ and that the number of clusters at the top level of the cluster hierarchy will not exceed $k_{top,max}$ to ensure the size of the sub-problems are manageable. However, as the underlying reason for establishing the sub-problems is to improve the overall computational runtime of solving the more complicated problem, the tests results presented in this section focus on the computational runtime of the clustering algorithm under input conditions that are representative of applying this algorithm to aid the solving of more complicated problems as indicated in [4.28].

The tests were run to vary $p_{c,max}$ = {3..12} at increments of 3, $k_{top,max}$ = {3..12} at increments of 3, and the number of points, p = {50-300} at 50 point increments. The points were randomly distributed within the intuitive 2D problem space in an 1200 by 800 area as iterative clustering algorithms like the one presented in this paper commonly take longer to settle in random environments where there is no structure to exploit. Hence, this test environment will provide more conservative results on the algorithms computational time performance.

For each ($p_{c,max}$, $k_{top,max}$, P) combination, 100 tests were run on a 2.0GHz PC with 2.0 GB of RAM which is a nearly equivalent machine to the ones used in previous studies that this paper's work is intended to help further [4.28]-[4.30]. As most of these previous studies were interested in real-time solutions to complicated problems in under 0.25-0.5 seconds, this paper's study focuses on the range of cases that this paper's algorithm can cluster in under 0.1-0.15 seconds so that the remaining time may be reserved for the solving the more complicated problem that the clusters summarize. The results of these tests are shown below in Table 4.1.

Table 4.1: Average Computational Runtime Varying |P|, $p_{c,max}$ and $k_{top,max}$

$p_{c,max}$	t	$k_{top,max}$= 3	$k_{top,max}$= 6	$k_{top,max}$= 9	$k_{top,max}$= 12
3	50	3.00	2.97	3.04	2.97
3	100	12.40	12.47	12.36	12.38
3	150	27.42	27.38	27.38	27.67
3	200	48.16	47.96	48.07	47.96
3	250	70.75	70.82	70.67	70.66
3	300	102.99	102.96	102.92	102.79
6	50	3.59	3.57	3.58	3.62
6	100	12.47	12.60	12.48	12.38
6	150	26.15	26.26	26.38	26.33
6	200	45.49	45.49	45.54	45.64
6	250	71.86	71.75	71.85	72.12
6	300	104.58	104.74	104.58	104.84
9	50	4.21	4.22	4.12	4.12
9	100	15.22	15.20	15.19	15.23
9	150	33.02	33.07	33.01	33.11
9	200	57.83	57.91	57.76	57.78
9	250	89.83	90.01	90.03	89.84
9	300	129.49	129.74	129.89	129.73
12	50	5.03	5.00	5.01	5.09
12	100	19.01	19.07	19.12	19.08
12	150	41.54	41.56	41.61	41.61
12	200	71.99	71.95	71.79	71.97
12	250	112.04	112.34	111.98	112.09
12	300	161.43	161.48	161.43	161.43

As to be expected, as the number of targets increases, so does the time in a non-linear fashion. This is graphically demonstrated in Figure 4.5 below, which clearly shows that the overall algorithm scaling is dominated by $|P|^2$ very closely, i.e. all shown lines fit to the general form of $c|P|^2$ with R-squared values of nearly 1, where the value of c is fairly small and varies with $p_{c,max}$ as will be discussed later.

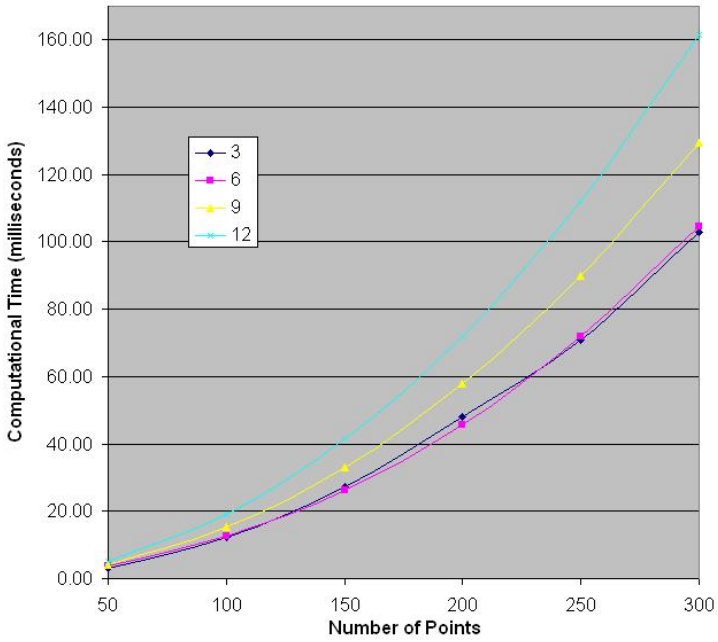

Figure 4.5: Average Computation Runtime vs. the Number of Points, |P|, for Values of $p_{c,max}$={3,6,9,12}

This can trend can be expected not only from the big "O" runtimes of the algorithm's components given in earlier sections but from the iterative and hierarchical nature of the algorithm which is nearly certain to induce non-linearities in any algorithm's scaling. The more interesting point however is that changing $k_{top,max}$ has practically no influence on the computation time as can bee seen by looking across the rows of Table 4.1. This is certainly true for the range of $k_{top,max}$ values that are of interest in this study largely because within this range of $k_{top,max}$ values there is little difference in the number of hierarchical levels to the final cluster solution. If $k_{top,max}$ is increased more significantly however, it can reduce the number of out iterative loops, i.e. the loop around Block 1-8 of Figure 4.1, and hence reduce the overall computation time. However as $k_{top,max}$ values greater than 12 are not of relevance to the direct

potential applications of this paper's work, this is beyond the scope of this paper's tests.

Although changing $k_{top,max}$ within the ranges of these tests does not influence the computational time, changing the limit of $p_{c,max}$ does have a significant influence that is best demonstrated in Figure 4.6. In Figure 4.6, the general shift in each line is caused by increasing the number points as is also shown in Table 4.1. However it can also bee seen that increasing $p_{c,max}$ also increases the computational time and that increasing $p_{c,max}$ alone, i.e. moving along any one line, also increases the time non-linearly.

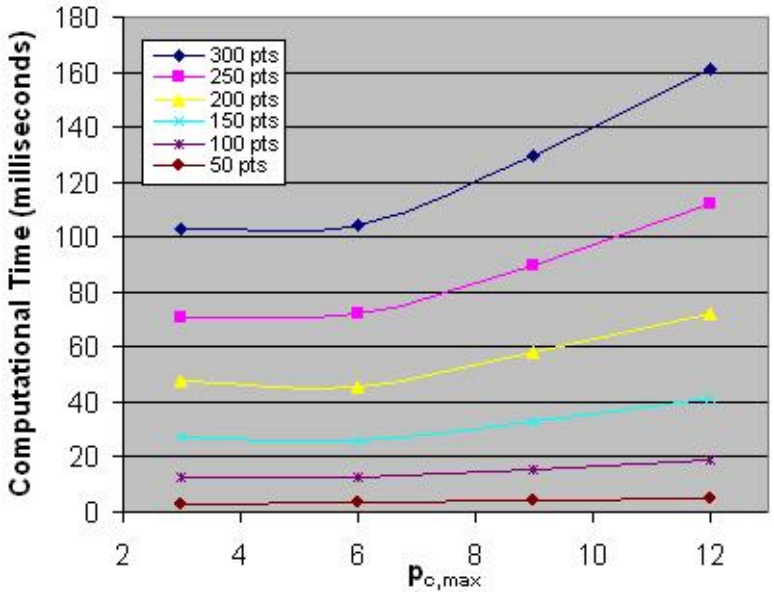

Figure 4.6: Average Computation Runtime vs. Maximum Number of Points Per Cluster, $p_{c,max}$, for the Point Sets of Size, |P| = {50,100,150,200,250,300}

At first this may appear counterintuitive since as $p_{c,max}$ increases the average height of the hierarchy and the average number of clusters both tend to decrease. Furthermore, when either the height or the number of clusters decreases, in general the time required to determine all cluster to cluster costs (or any other cluster characteristics) also decreases. However, as the size of an allowable cluster increases, the time required to exit the assignment/splitting/merging iterations of Figure 4.1, Blocks 3-7 increases as would the time to determine any individual cluster characteristics such as the cluster's near-centroid. It is the later increase in time from Figure 4.1, Blocks 3-7 that is the dominant factor however and it is once again the iterative nature of Figure 4.1, Blocks 3-7 that causes the non-linearity.

Even though these non-linearities exist, they are quite common in clustering algorithms as most do involve some iterative element that is fundamental to the algorithm. In addition, the times reported in Table 4.1 are almost all below the study's goal of 0.15 seconds. The only cases that are slightly above are the 300 point, $p_{c,max}$ = 12 case which is at an extreme of the study. The standard deviations of the average computation times of Table 4.1 were also very good and never exceed 10% of the average. Furthermore, it was verified that the condition requirements of $p_{c,max}$, $k_{top,max}$, and $D_{max,\alpha}$ were all met in all tests demonstrating that this clustering algorithm met all of this study's goals very well.

4.10 Conclusions

The new clustering method presented in this paper has been shown as a fast technique for partitioning large MTSP NP-Hard problems into a hierarchy of cluster sub-problems. The new method has also been shown to guarantee both the size of each cluster sub-problem and the size of the top of the hierarchy to be within user specified

input bounds in order to provide control on the scale the sub-problems. Furthermore, this method has also been shown to be able to function with a minimal set of required inputs and provides rules for establishing internal parameters in the lack of additional problem information. Splitting and merging cluster adaptation phase rules as well as the concept of the near-centroid were also introduced in establishing this paper's clustering method as a viable option for a wider variety of real-world applications. Finally, as the primary purpose of this new method is to assist in the computation time of solving NP-Hard problems, the speed of this new method was also verified through a series of implementation tests under conditions representative of MTSP NP-Hard problems.

REFERENCES

[1] Gerkey, B., and Mataric, M., "A formal analysis and taxonomy of task allocation in multi-robot systems" *International. Journal of Robotics Research* 23(9):939-954, September 2004

[2] Campbell, M., "Planning Algorithm for Multiple Satellite Clusters," *Journal of Guidance, Control and Dynamics*, Sept-Oct 2003.

[3] G. Thomas, A. M. Howard, A. B. Williams, and A. Moore-Alston, "Multi-robot task allocation in lunar mission construction scenarios," in *IEEE International Conference on Systems,* Oct 2005

[4] Chandler, P., Pachter, M., *et al.* "Distributed Control for Multiple UAVs with Strongly Coupled Tasks," AIAA *Guidance, Navigation, and Control Conference*, August 2003

[5] Richards, A., Kuwata, Y., How, J., "Experimental Demonstrations of Real Time MILP Control" *AIAA Guidance Navigation and Control Conference*, Aug. 2003.

[6] Y. Kuwata, A. Richards, T. Schouwenaars, and J.How, "Distributed Robust Receding Horizon Control for Multi-vehicle Guidance" *IEEE Transactions on Control Systems Technology Journal*, 2007

[7] Purwin O., D'Andrea R.: "Cornell Big Red 2003", in: Polani D., Bonarini A., Browning B., Yoshida K. (Eds), *Robocup 2003: Robot Soccer World CupVII, Lecture Notes in Artificial Intelligence,* Springer, Berlin, 2003

[8] M.B. Dias, R.M. Zlot, N. Kalra, and A. Stenz, "Market-Based Multirobot Coordination: A Survey and Analysis," *Proceedings of the IEEE,* Vol. 94, No. 7, July 2006

[9] Davidson, I. and Ravi ,S.S. "Agglomerative Hierarchical Clustering with Constraints: Theoretical and Empirical Results" *European Conference on Principles and Practice of Knowledge Discovery*, Nov. 2005

[10] Nanni , M., "Speeding-Up Hierarchical Agglomerative Clustering in Presence of Expensive Metrics" *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2005

[11] Zho,Y. and Karypis,G. "Hierarchical Clustering Algorithms for Document Datasets", *Data Mining and Knowledge Discovery*, Vol. 10 No. 2, March 2005, pp. 141–168

[12] Ait Kbir, M., Maalmi, K., Benslimane, R., and Benkirane, H., “Hierarchical Fuzzy Partition for Pattern Classification with Fuzzy if-then Rules” *Pattern Recognition Letters*, v.21 n.6-7, p.503-509, June 2000

[13] Montes, J.C., Llorca, R.M., and Grau, J.P., "Building Interpretable Fuzzy Systems: a New Approach to Fuzzy Modeling," *Electronics, Robotics and Automotive Mechanics Conference*, pp.117-122, 2006

[14] Tsekouras, G.E., and Sarimveisb, H., “A New Approach for Measuring the Validity of the Fuzzy C-Means Algorithm” *Advances in Engineering Software* Volume 35, Issues 8-9, pp.567-575, 2004

[15] Hoppner, F., and Klawonn, F., “A New Approach to Fuzzy Partitioning” *IFSA World Congress and 20th NAFIPS International Conference*, 2001

[16] Honda, K., and Ichihashi, H., “A New Approach to Fuzzification of Memberships in Cluster Analysis” *Modeling Decisions for Artificial Intelligence*, pp.172-182, 2005

[17] Shi, J., and Malik, J., "Normalized Cuts and Image Segmentation",*IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 888-905, August 2000

[18] Meila, M., and Shi, J., "Learning Segmentation with Random Walk", *Neural Information Processing Systems*, 2001

[19] Kannan, R., S. Vempala, and A. Vetta, "On Clusterings - Good, Bad and Spectral", *Yale University Technical Report*, 2000

[20] Ng, A. Y., M. I. Jordan, and Y. Weiss "On Spectral Clustering: Analysis and an algorithm", *Advances in Neural Information Processing Systems 14*, 2001

[21] Weiss, Y. "Segmentation using eigenvectors: a unifying view", UC Berkeley Technical Report, 1999

[22] Ayad , H. and Kamel, M., “Finding Natural Clusters Using Multiclusterer Combiner Based on Shared Nearest Neighbors,” *Proc. of the Fourth Int’l Workshop on Multiple Classifier Systems*, 2003.

[23] Kuncheva,L.I., and Vetrov, D.P., “Evaluation of Stability of k-Means Cluster Ensembles with Respect to Random Initialization” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 28, No. 11, Nov. 2006

[24] Topchy, A., Jain,A.K., and Punch,W., “Combining Multiple Weak Clusterings,” *Proc. IEEE Int’l Conf. Data Mining,* pp. 331-338, 2003

[25] Fred ,A., and Jain,A.K., "Robust Data Clustering," *Proc. IEEE CS Conf. Computer Vision and Pattern Recognition*, 2003.

[26] Fern, X.Z. and Brodley, C.E., "Random Projection for High Dimensional Data Clustering: A Cluster Ensemble Approach," *Proc. 20th Int'l Conf. Machine Learning*, pp. 186-193, 2003.

[27] Greene, D., Tsymbal, A., Bolshakova , N., and Cunningham, P., "Ensemble Clustering in Medical Diagnostics," Trinity College Technical Report, Dublin, Ireland, 2004.

[28] D. Schneider and M.Campbell "Improved Optimistic Predictive Cost Method For Faster G*TA Real-Time Task Allocation" *Paper submitted to IEEE Transactions on Robotics*

[29] Squire P.N., Galster, S.M., & Parasuraman, R. "The effects of levels of automation in the human control of multiple robots in the RoboFlag simulation environment." *Proceedings of the Second Human Performance, Situation Awareness, and Automation Conference*, 2004

[30] Campbell, M., D'Andrea, R., Schneider, D., Chaudhry, A., Waydo, S., Sullivan, J., Veverka, J., Klochko, A., "RoboFlag Games using Systems Based, Hierarchical Control," *American Control Conference*, June 2003.

CHAPTER 5

REAL-TIME, LARGE SCALE TASK ALLOCATION APPROXIMATION USING HIERARCHICAL CLUSTERING AND G^*_{TA}

One of the most significant challenges facing NP-Hard task allocation research is the ability to develop methods that can provide real-time solutions to large scale problems. This paper offers a new approximation method, based upon the fast optimal G^*_{TA} task allocation method, that can solve problems on the order of hundreds of tasks for groups of agents ranging is size from 2-6, in computation times that are up to two orders of magnitude faster than current standard approximation methods. Furthermore these results are achieved while still offering solutions of comparable quality to those standard methods. The premise of the new method, called H-G^*_{TA}, is that since these task allocation problems are NP-Hard, faster solutions can be obtained by partitioning large problems into a hierarchy of smaller sub-problems that can be solved optimally within a reasonable amount of time. Details on the employed adaptive K-means partitioning technique, the incorporation of G^*_{TA} to solve sub-problems, the combining of the sub-problem solutions to obtain a final solution to the original problem, and on applying a K-Opt post-optimization technique are provided throughout the paper. A series of conservative implementation tests are also presented which validate the computational runtime and solution quality of the H-G^*_{TA} method.

5.1 Motivation

The strong need for developing multi-agent task allocation systems that can be effectively applied in real-time scenarios can be seen in an ever growing variety of

applications that range from space exploration [5.1],[5.2], to coordinated UAV control [5.3],[5.4] to even robotic soccer [5.5]. However, this need is perhaps most significantly shown in the amount of research that has devoted to this area. A thorough description and taxonomy of the variety of problems addressed and some of the main approaches used to solve these problems were conducted by the Stanford / USC [5.6]. However due to the NP-Hard nature of many of these task allocation problem variations, one of the largest challenges that continues to be a main topic of research is the scalability of these algorithms. In order to be applicable to real-world scenarios many studies limit their problems to be on the order of 10-50 tasks for even approximation methods. This paper offers a new hierarchical method, H-G^*_{TA} that will be shown to be viable for handling up to 6 agents, 250 task problems while producing comparable solution quality to a current standard approximation method in up to 2 orders of magnitude faster computation times.

Being able to solve problems of this larger scale with reasonable efficiency has become an important matter in recent applications such as the NASA/NOAA OASIS project where unmanned boats must survey highly dynamic storm and hurricane environments [5.7]. The importance of similar missions, particularly with regards to UAV control, has been has been formally recognized by the U.S. Department of Defense [5.8].

In developing a method to handle this scale problem, both current optimal and approximation algorithms were investigated. Optimal methods that are largely based on Mixed Integer Linear Programming (MILP) techniques have been used extensively within such groups as MIT, Wright Patterson AFRL, Berkley and several other leading universities [5.2],[5.9]-[5.17] to the point where Dr. Sangiovanni-Vincentelli's

group at Berkeley has even published a classification of MILP representations of the problem [5.15]. In general however, most of these methods were far too slow and memory resource intensive to be effective for the highly dynamic environments of interest. Recent advancements however have allowed the relatively small scale problems of up to approximately 10 tasks that are typical of NP-Hard task allocation problems, to be solved optimally in times as fast as some current approximation methods. The most notable of which is perhaps the development of the optimal G^*_{TA} method which was shown to run on average 2 orders of magnitude faster than a traditional MILP implementation [5.18]. However, even with these improvements, optimal methods are still too computationally slow to handle problems of this scale in rapidly changing environments.

Due in part to these significant solution time requirements, the other primary approach for solving task allocation problems are approximation heuristic methods. These methods sacrifice the guarantee of optimality in return for faster computation times. This area of research in particular has been growing very quickly, particularly for less complex versions of task allocation problems. Many notable recent methods make use of a variety of algorithms from other fields including negotiation and even financial models as well and have been implemented by groups at Carnegie Mellon, Stanford, USC and LAAS-CNRS [5.19]-[5.24]. Due to the popularity of approximation methods, several survey papers have been written to summarize the research community's efforts in this area which include the Gerkey and Mataric [5.6] and more recently the Dias, Zlot, Kalra, and Stenz paper from Carnegie Mellon paper which focuses on summarizing the fast growing area of market-based approaches [5.24].

Recent work has tried to overcome the optimal computation time issues and the approximation method's lack of accuracy through combining the two to create more reliable approximations. One example of this is the coordinated reconnaissance work of Ousingsawat & Campbell which combined MILP with clustering techniques to reduce the problem size [5.25]. Another example is the intercept path planning work done by Earl & D'Andrea which used MILP not to determine optimal assignment but whether there existed any assignment that met a certain goal [5.16]. Most recently, Rathinam & Sengupta from Berkley have modified Held-Karp's lower bounds TSP method to handle the multiple depot, multiple salesman task allocation problem to a 2-approximation LP-relaxation method that forces fixed ending tasks [5.26]. Some of the most notable work though has been done under Dr. John How at MIT, including a study of MILP experimentations where MILP is used to solve higher level problems at a lower frequency (on the order of seconds) and model predictive control (MPC) is used in between MILP based updates [5.10],[5.17].

The work presented in this paper also utilizes a combined approach to create a new method, H-G^*_{TA}, that integrates the hierarchical clustering method of [5.27] with the optimal G^*_{TA} method of [5.18]. Together these algorithms are respectively employed to organize the initial large NP-Hard problem into a structured set of smaller sub-problem clusters that can be solved optimally within a reasonable amount of time. In addition to effectively integrating these algorithms, the new work presented in this paper also provides algorithms for combining the results of the sub-problems to develop an overall summary problem and then for recursively "backing out" a final solution to the original problem. This paper also discusses the use of an inter-cluster characteristic, which will be referred to as clusters' neighbor pairs, that is based on the

concept of single linkage clustering to provide a useful mechanism for determining relationships between sub-problems.

The paper begins with first defining the task allocation problem variation focused on in this study in Section 5.2 and relates this variation to the taxonomy offered by [5.6]. Then in Section 5.3, the motivation and logic behind the design of the H-G^*_{TA} is presented along with a walkthrough of its core algorithmic components. The following sections then address each of these components in the order they occur in the overall H-G^*_{TA} method. Section 5.4 describes the incorporation of the hierarchical clustering method presented in [5.27]. Particular emphasis is given to this algorithm's role in fixing both the maximum scale of any sub-problem as well as the size of the hierarchy's top summary problem in order to control the computation time in later solving these smaller problems optimally. This section also describes how the near-centroid option of the hierarchical clustering method was implemented to allow the H-G^*_{TA} method to be more widely applicable to a variety of problems and to provide a more conservative estimate of H-G^*_{TA}'s performance.

Section 5.5 then offers two alternative options for solving the sub-problems. The first is a more traditional approach to cluster-based algorithms that places a strong emphasis on the clusters' centroids. The second alternative first introduces the "neighbor pair" inter-cluster relationship characteristic and then demonstrates how this can be utilized in solving the sub-problems as well. This section also provides a brief comparison of the two alternatives. With the clustering and sub-problem development discussed, Section 5.6 describes how the hierarchy of these cluster sub-problems culminate into a top level summary problem that is also guaranteed to be of a scale that can be solved optimally within a reasonable amount of time.

Section 5.7 then provides a description of two alternatives for recursively determining the final solution to the original problem from the solution to the summary problem. Once again, the first alternative follows the traditional approach of placing significant algorithmic value on the centroid while the second alternative is focused on exploiting the advantages that the neighbor pair characterization can provide. The two alternatives are briefly compared in this section, but the majority of this discussion is left for presentation of the implementation test results in Section 5.11.

Section 5.8 also provides a discussion on the incorporation of applying a post-optimization method to the final solutions found from the recursion methods presented in Section 5.7. This section focuses particularly on the K-Opt method while using small values of "K". Section 5.9 then provides a detailed description of the greedy comparison method used in the final implementation tests. This section also provides justification for incorporating this method as the comparative method and relates its performance to the currently popular area of market-based approximation methods.

Section 5.10 then provides details on the set-up of the implementation tests. This is then followed in Section 5.11 with a discussion of the tests' results and the emerging trends from the data which show the H-G^*_{TA} as a new fast and viable method for solving very large scale NP-Hard task allocation problems. These trends are in turn supported in this section by a detailed description of the runtime analysis of the H-G^*_{TA}'s main algorithmic components as they were introduced throughout this paper.

5.2 Task Allocation Problem Definition

The task allocation problem defined in this paper is the same as the one defined in the previous work [5.18]. In general, "task allocation" is used to describe a variety of problems where there is a given set of tasks (a.k.a. targets, goals, etc) that must be performed, and there is a given set of agents (a.k.a. robots, sources, etc) that can perform the tasks. This paper considers the common form where any task may be assigned to any source and any source may be assigned multiple tasks or none at all.

In the problems considered here, there is a different assignment cost (or utility) for each task/source pair as well as a different assignment cost for transitioning from any one task to another. This last statement connotes that the order in which tasks are assigned to a source does influence the overall cost. In addition, all costs are assumed to be non-negative and the cost associated from transitioning from state A to state B does not have to be the same as the cost associated from transitioning from state B to state A. The problem definition is further generalized to allow it to be more applicable to more real world scenarios by allowing the triangle inequality not to hold. This last point is a generality that is not always commonly handled in task allocation algorithms and will be highlighted later in the discussion.

For completeness, the task allocation algorithms presented in this paper also allow that every task assignment may have an associated list of constraints. These constraints may be either source specific, such as a limit on the amount of a resource that each source can expend, or task specific such as requiring a set resource amount or specific time constraints. These constraints are also allowed to vary depending upon the previous task assignments, i.e. there can be resource expenditures for transitions between tasks. Constraints are also allowed on the problem as a whole, such as

requiring that the total resource expenditure across all sources does not exceed a given limit. However this paper only considers problems where there does exist at least one feasible solution.

The topic of constraints in task allocation problems is actually quite involved and therefore this paper focuses itself to problems where the constraints do not influence the solution to the problem but rather the solutions are strictly cost driven. However as the above mentioned constraints appear in many applications, the algorithms presented in this paper designate how constraint checks are incorporated.

Given the sources, tasks, and the calculated costs and resource / constraint requirements, the goal of the task allocation problem is then to determine which tasks should be assigned to which sources, and equally importantly, in which order those tasks should be assigned, so as to assign all tasks while incurring the overall minimal cost without violating any constraints.

Stated more formally, let s stand for the number of agents (or sources) that can be assigned tasks and let t stand for the number of tasks (or targets) that must be assigned to the sources. Let the means of transitioning a source from its initial state to a target's state and likewise the means for transitioning from one target state to another, be referred to as a "path". Additionally, a path is also assumed to contain all associated assignment and state transitioning costs and constraint information. Finally, let the ordered set of paths assigned to a source be referred to as a "trip", where Tr_i stands for the trip of source i. As a result, all trips always begin with a source and then contain an ordered list of the targets assigned to that source.

Table 5.1: Summary of Key Problem Definition Terms

s	Number of sources
t	Number of targets
path	Means of transitioning for a source from its initial state to a target state or from one target state to another
trip	Ordered set of paths to be determined for each source, where Tr_i stands for the trip of source i

In this paper, the symbol "V" is used to represent the set of all vertices, "S" is the subset of source vertices and "T" the subset of target vertices, which contain "v", "s", and "t" members each respectively as shown in the equations below.

$$S \subset V, \quad T \subset V, \quad s.t. \forall V_i \in V \ \ if (V_i \in S) \Rightarrow (v_i \notin T) \tag{5.1}$$

$$v = s + t \tag{5.2}$$

With these terms in place, the task allocation problem can be defined as: Given s sources, t targets and a set of all available paths, the solution method must create a set of trips for the sources, Tr_i for all $i \in S$, such that each target is assigned to or "visited by" at least one source, without violating any constraints. Furthermore, this assignment must result in the minimal overall combined cost of all of the sources' trips, as expressed in (5.3), where J is the overall combined cost and $J_i(Tr_i)$ is the cost incurred from the trip of source i. It is also assumed that a "one way trip" is the default problem variation where a source may end its trip at any of the targets or remain at its current state without being assigned any targets at all.

$$J = \sum_{i=1}^{s} (J_i(Tr_i)) \tag{5.3}$$

Defining the problem in this manner allows the problem to be interpreted as a graph problem as shown in Figure 5.1. Using the graph representation the input was standardized as:

1. A highly connected directional graph where each vertex, $v_i = [x_i, y_i]$ is defined as either a source or a target and each graph edge, $e_{ij}(v_i, v_j)$, is assigned a weight, $w_{i,j}$, equal to the cost of the path between vertices v_i and v_j.

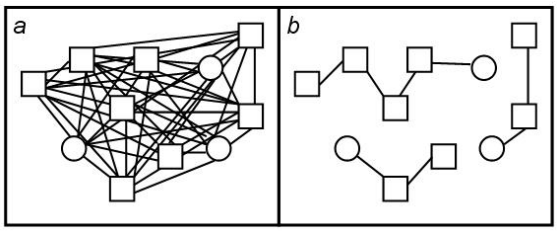

Figure 5.1: Graphical Representation of the Task Allocation Problem with sources as circles and targets as squares a) input as a highly connected graph b) solution as series of selected graph edges to form 3 "trips", Tr_i, $i \in \{1..3\}$

Similarly, the output solution was standardized as:

1. The final cost associated with the solution, *J*.
2. The trip that should be executed for each source, Tr_i, $i \in \{1..s\}$.
3. A report on the constraints and/or resource usage. For this paper's tests, the constraints measured include the amount of fuel used, the overall time it would take for the sources to execute the solution, and the time that each individual task was completed where again the constraint / resource usage information is provided via the path input information.

4. The solution method's total computation time

In terms of [5.6]'s taxonomy, this described problem is a more difficult version of the Single Task Robots – Single Robot Tasks – Time Extended Assignment or ST-SR-TA case, i.e. single task robots implies that a robot can handle several ordered tasks but only one at a time, and single robot tasks implies that tasks only require a single robot's attention at some point in the task allocation. A fairly well known example of a ST-SR-TA problem is the ALLIANCE Efficiency Problem (AEP) first stated by Parker [5.28]. The problem presented here is a more difficult ST-SR-TA is because of the interdependency between the costs, i.e. whether a robot visits target A or target B first influences the cost to travel next to a target C. With this observation, it becomes apparent that the problem presented here is an instance of the Multiple Depot, Multiple Traveling Salesman Problem and is hence NP-Hard, as was shown by Korte & Vygen in 2000 [5.29]. As the Multiple Depot, Multiple Traveling Salesman Problem description has perhaps the most intuitive meaning, this paper will henceforth refer to any problem or sub-problem fitting this description as an MTSP.

As discussed in Section 5.2, both optimal and approximation methods [5.7]-[5.25] have been applied to similar variations, and therefore this is a good problem to demonstrate the abilities of the new H-G^*_{TA} method. This paper's study will show H-G^*_{TA} as a new method to solve the task allocation MTSP described in this section for very large sets of targets as compared to current standard approximation methods, in computational runtimes that are faster than current standard methods while providing a comparable approximation of the optimal solution.

5.3 Hierarchical G^*_{TA} Overview

The basis of the new hierarchical G^*_{TA}, H-G^*_{TA}, stems from the NP-Hard exponentially scaling nature of MTSP problems. As was shown in [5.30], smaller scale problems around 6 sources and 6 targets, on average G^*_{TA} was able to optimally solve the problems faster than 250Hz on a modest PC machine. However, even though these results showed in all cases that the new G^*_{TA} method significantly improved the average runtime to produce guaranteed optimal solutions over MILP based methods, increasing the size of the problem by even a few targets could increase the average computation runtime of either method by more than an order of magnitude.

Hence, given that small problems can be solved fairly quickly, a hierarchical G^*_{TA} method, which will be denoted as H-G^*_{TA}, was designed to solve large scale MTSP problems. The following steps outline the solution process of the H-G^*TA method with the aid of the block diagram of Figure 5.2 below.

1. Break the original large problem into smaller sub-problems no larger than 6 sources and 6 targets, so that the sub-problems can be solved in a reasonable amount of time, but still represent the characteristics of the original problem. This sub-problem creation is performed via the hierarchical clustering of the target set *T*, through the method presented in [5.27]. The set of clusters created at level α of the hierarchy is referred to as K_α and may be run through the optional steps of $k_{top,max}$ limit forcing and establishing neighbor pairs which will be discussed in Sections 5.4 and 5.5.
2. Summarize each cluster sub-problem of K_α in terms of the cost and resource usage that would result from assigning the entire cluster to a single source. This sub-

cluster summarization is handled through solving a TSP of each cluster's members by the methods discussed in Section 5.5.

3. Summarize the smaller sub-problems at the top level of the hierarchy so that the summary corresponds to the larger original problem as accurately as possible. The set of clusters at the top of the hierarchy will be referred to as K_{top}.
4. Create a small scale MTSP at the top level of the hierarchy that considers only the input source set, S, as the sources and the top level of clusters, K_{top}, as the targets. This top level MTSP will be referred to as the Summary problem and presented more formally in Section 5.6
5. Solve the Summary problem in a quick but accurate manner via the G^*_{TA} method. Each source's trip in the solution to this Summary problem will be referred to as $Tr_{i,top}$, and each $Tr_{i,top}$ consists of the source i, and a subset of the top level clusters, K_{top}.
6. Using the Summary problem solution, recursively reassemble the smaller sub-problems solutions to create a solution to the original large problem via the methods described in Section 5.7. Each source's trip within the solution to the original problem will be referred to as $Tr_{i,final}$ and the trip for that source i consists only of that source and members of the target set, T.
7. Optionally review and improve upon the reassembled solution to the original large problem using a post-optimization technique. The technique that will be utilized in this paper is the K-Opt method, as discussed in section 5.8.

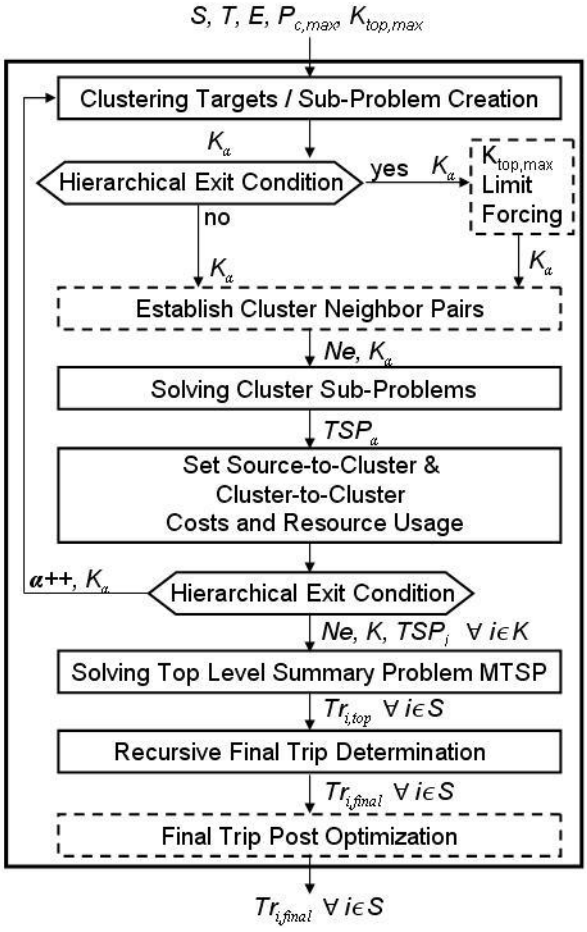

Figure 5.2: Algorithmic Flowchart for the H-G^*_{TA} Method

5.4 Sub-Problem Representation through Target Clustering

A primary goal of this paper is to develop the H-G^*_{TA} method to be able to address problems with up to hundreds of targets. However, the optimal G^*_{TA} method has only been shown to handle around 6 targets in the desired computation time range. Therefore, the focus of this first section of the H-G^*_{TA} method is to break the original large problem into a series of sub problems that can be solved very quickly but still represent the characteristics of the original problem.

The partitioning method chosen for this undertaking is the hierarchical adaptive K-means clustering technique presented in [5.27]. In describing this clustering technique as applied in the H-G^*_{TA} method more formally, let K stand for the total set of all of clusters in the hierarchy; let K_α stand for the set of clusters formed at the cluster hierarchy level α; and let K_{top} stand for the set of clusters formed at the top level of the cluster hierarchy. Similarly let k_α and k_{top} stand for the number of clusters in set K_α, and K_{top} respectively. (should I delete "let… " as I mentioned most of this before)

In short, the clustering technique of [5.27] is applied to create a cluster hierarchy out of the target set, T, only. At the bottom level, the targets are clustered according to their transition cost, i.e. the weight of each target to target edge, as shown in Figure 5.3. Then to form the subsequent higher level, each cluster centroid (or near-centroid to be precise in the terms of [5.27]) is treated as a new target and these "centroid targets" are then clustered themselves to form the next level as shown in Figure 5.3b. The cost from centroid to centroid is specified according to the method for determining the paths listed in Section 5.2 [5.18]. This process of interpreting the most recent cluster level's centroids as targets for the next level of clustering is repeated until the top cluster level consists of a small enough number of clusters that can be solved in a reasonable amount of time as the summary problem as discussed in detail in Section 5.6.

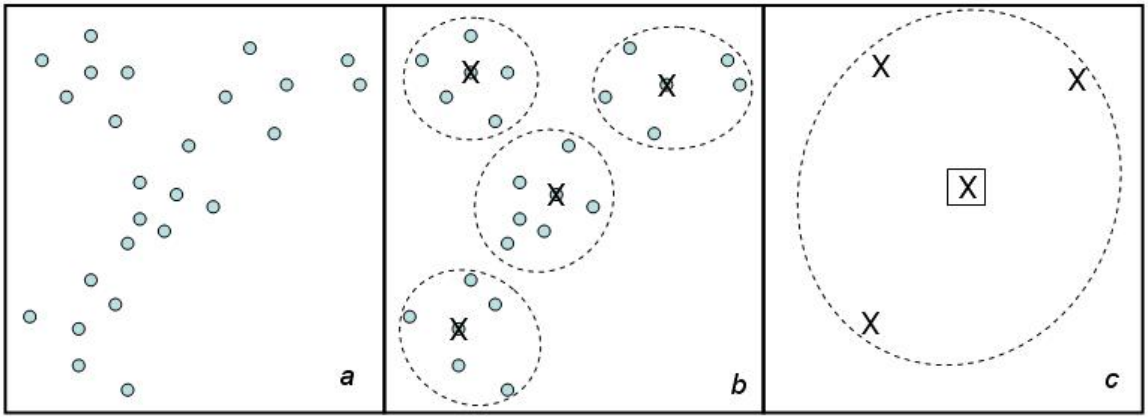

Figure 5.3: a) Set of targets, T b) clustering of T; X's represent the near-centroid of each cluster c) near-centroids of the first level of clustering treated as targets for the next level; the box represents the near-centroid of the next level

By performing this clustering, the problem has been reduced to solving the K set sub-problems and the one summary problem instead of solving one computationally-impractical large problem. The benefit of using specifically the clustering technique in [5.27] however is that the size of each cluster sub-problem and the summary problem are also limited to within the clustering technique's inputs.

The inputs to the clustering technique are simply the set of points to be clustered, i.e. the target set T, and two other input parameters $p_{c,max}$, $k_{top,max}$. The first parameter $p_{c,max}$ sets the maximum number of points allowed in any cluster and $k_{top,max}$ sets the maximum number of clusters that are allowed at the top level of the cluster hierarchy. By setting $p_{c,max}$ and $k_{top,max}$ to be values that the previous work in [5.18],[5.30] showed to produce average computational runtimes that are acceptable, the sub-problem and summary problem runtimes can also be controlled to acceptable limits.

Although it will become clearer in Section 5.5 which deals with the methods used to solve the sub-problems and summary problems, as a general rule, $p_{c,max}$ and $k_{top,max}$ should both be set to the maximum allowable values that will permit the entire H-G$^*_{TA}$ method to run within an acceptable time. In general, larger values of $p_{c,max}$ allow the clustering method to develop a better representation of the original problem.[5.27] Similarly as with nearly all hierarchical methods, the taller the hierarchy and the smaller the top level, the poorer the representation of the original problem. Although the height of the hierarchy is determined by the a combination of $p_{c,max}$, $k_{top,max}$, $|T|$, and the distribution of T across the problem space, the size of the top cluster level can be forced to be exactly equal to $k_{top,max}$. This is an option of the [5.27] clustering technique and but forcing k_{top} to equal $k_{top,max}$ is a standard within the H-G$^*_{TA}$ method as long as $k_{top} < |T|$. For completeness, in the trivial cases where $k_{top} \geq T$, no clustering will occur and the standard G$^*_{TA}$ method will be run. Fixing the size of the top level summary problem also has the benefit of reducing the standard deviation of both the computation time and the percent error results as this is one less source of variance.

$$k_{top} = k_{top,\max} \tag{5.4}$$

It should also be noted that the [5.27] rules for applying an optional cluster cost distortion max, D_{max}, criteria were also included in the H-G$^*_{TA}$ implementation as it was highly recommended by [5.27]. In addition, the near-centroid option of [5.27] was also incorporated. This option states that the centroid chosen for a cluster does not necessarily have to be the perfect average of the properties of its members, but rather the cluster member that is closest to that average.

The near-centroid option, as shown in Figure 5.3b and 5.3c, assures that cluster's centroid is a feasible state in the problem space as it directly relates to one of the original targets. This option however requires that an external method for determining a representational target from a set of given targets is available. In many problem cases, this can be a very quick calculation. For example, in the intuitive case of an MSTP in a 2D space where the cost is simply the distance between targets, the near-centroid is simply the closest target to the average of all of the cluster's targets. Furthermore in this specific case, the near-centroid option may not even need to be applied.

Specific applications of the H-G$^*_{TA}$ may be able to relax both the D_{max} and near-centroid requirements. As is implied in [5.27], dropping the near-centroid requirement when possible is also recommended as a "true" centroid when available is typically a better representation of a cluster. Hence, using a true centroid can potentially achieve improved performance over the results presented in Section 5.11. However the near-centroid option was employed in all of this paper's experimental studies as this paper is aimed at presenting a conservative view of the H-G$^*_{TA}$ method's performance and the near-centroid option allows the paper to demonstrate a "worst-case" performance of the most generally applicable version of the new H-G$^*_{TA}$ method.

5.5 Solving Cluster Sub-Problems and Establishing Clusters' Neighbor Pair Members

The bottom level of clustering discussed in the previous section follows a fairly classical approach. This section, however, describes how once this set of clusters is established, how the cluster sub-problems are solved and how the sub-problem

information is used in creating clusters of the clusters, which will be referred to as super clusters. There are two methods presented in this section for performing these tasks. The first method is a more traditional way of summarizing cluster information and will be referred to as centroid based.

The second method introduces a new concept for using clusters in MTSPs that is closely related to single linkage clustering, i.e. determining the lowest cost edge between two clusters. As discussed in Section 5.5.3, the two vertices of this edge are referred to as the cluster's neighbor members, and hence the second method will be referred to as neighbor based. Section 5.5.5 also provides a brief discussion on variants of the neighbor based method and justification for why the neighbor based method of Section 5.5.3 is employed instead.

5.5.1 Centroid Based Cluster to Cluster Transition Costs

The centroid based method is the simplest option of those discussed throughout Section 5.5 and hence was developed to offer the fastest option as well. This method represents the cost between clusters as being the cost between centroids. In using the near-centroid option, the near-centroids at any level of the hierarchy relate back to the exact state of a target. Hence, as the target to target costs have already been calculated, they need only be copied as the cluster to cluster costs. If the true centroid method, mentioned in Section 5.4, is able to be used, this centroid based method would require the centroid to centroid costs to be calculated by the same manner that the target to target costs were calculated. Although application dependant, typically this is very fast, especially with respect to the rest of the algorithm.

5.5.2 Centroid Based Cluster Sub-Problems

In order for the summary MTSP problem of the sources and the top level clusters' centroid to better represent the entire original problem, each cluster should contain information estimating the cost and resource usage of assigning all of its sub-clusters down to the bottom clusters of targets. In order to obtain this estimate, the centroid based method calculates a single TSP using the centroid as the source and the rest of the cluster's members as targets. At this point, as it is unknown what the final trips in the solution to the original problem will be, there is no way of knowing from what previous target will a cluster be "entered into" or to what target the trip will go onto in the next cluster upon "leaving" a cluster. Therefore, solving the cluster TSP from the centroid, as the most representational point of the cluster, and ending at whatever member within the cluster results in the lowest cost TSP solution is reasonable way to obtain a cost and resource usage estimate.

As the clusters and hence these TSP problems are able to be limited in size to being no more than $p_{c,max}$ targets, it can assured that these sub-problems can be solved fairly quickly. Due to this problem size control, the H-G^*_{TA} method employs the original G^*_{TA} method to solve these TSPs optimally. In fact, if $p_{c,max}$ is of a small enough scale, the G^*_{TA} method is able to solve the TSP optimally nearly as fast as many approximation methods. However, as the solution to the TSP is only an estimate, using an optimal method for very small values of $p_{c,max}$ may not always improve the estimate that greatly. Specific values used for $p_{c,max}$ in this paper are shown in Section 5.10, however [5.18] shows that G^*_{TA} runtimes for optimally solving TSPs of 9 targets averaged just over 1 millisecond and for TSPs of 6 targets, the TSP could be optimally solved on average in less than a quarter of a millisecond.

In the special case, where the cost and resource usage to transition between any task A to any task B may vary depending upon the source, commonly referred to as a heterogeneous source case, each TSP may have to be solved separately for each source in order to provide the required sub-problem estimates. However if these costs, J_i , and resources, $r_{i,j}$, vary according to a source specific constant, β_i and $\gamma_{i,j}$, for all $i \in S$ and $j \in R$ as shown in equations (5.5) and (5.6), each cluster TSP can still only be solved once. As equations (5.5) and (5.6) are commonly reasonable assumptions, especially for the applications discussed in Section 5.1, only one TSP was solved for all cluster sub-problems while using the centroid based option.

$$\beta_1 J_1 = \beta_2 J_2 = \beta_3 J_3 = ... = \beta_i J_i \qquad \forall i \in S \tag{5.5}$$

$$\left\{ \begin{array}{l} \gamma_{1,1} r_{1,1} = \gamma_{2,1} r_{2,1} = \gamma_{3,1} r_{3,1} = ... = \gamma_{i,1} r_{i,1} \\ ... \\ \gamma_{1,j} r_{1,j} = \gamma_{2,j} r_{2,j} = \gamma_{3,j} r_{3,j} = ... = \gamma_{i,j} r_{i,j} \end{array} \right\} \qquad \forall i \in S, \forall j \in R \tag{5.6}$$

5.5.3 Neighbor Based Cluster to Cluster Transition Costs

Placing a strong emphasis on the centroid as a key summary point has been a traditional way of interpreting clusters. However, even if the cost between cluster's centroid is easily obtainable, more information regarding the relationship between clusters can be determined by investigating the clusters' member sets.

This section introduces a concept referred to as clusters' neighbor member pairs that is based on single linkage clustering. The neighbor member pair, *Ne*(A,B), from arbitrary cluster A and to arbitrary cluster B, where B != A, is defined as the member of cluster A and the member of cluster B who share the smallest cost outgoing edge from cluster A into cluster B. This is expressed in equation (5.7) and

shown in Figure 5.4. As the edges between any two targets are allowed to be directional, the best neighbor pair going from cluster A to cluster B does not have to be the best neighbor pair going from cluster B to cluster A.

$$Ne(A,B) = \min_{E}\{w_{i,j} : i \in A, j \in B, A != B\} \tag{5.7}$$

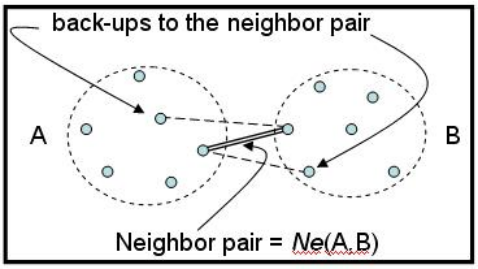

Figure 5.4: Establishing *Ne*(A,B), the neighbor pair of clusters A and B, and the back-up edges to that neighbor pair

Establishing the neighbor pairs can be a very useful characterization for cluster relationships especially in considering MTSP problems where it is highly conceivable that the solution may require a source to "travel' from one cluster to another. In this case, the neighbor pair not only provides information on the minimal cost and resource usage required to travel between two clusters but it also tells which cluster members may be the best to travel to and from. This is not to say that the edge between the neighbor pair cluster members is guaranteed to be part of the optimal final solution, as an exception will be shown later in Section 5.5.4 with Figure 5.5, but rather the neighbor pair provides a intelligent, intuitive estimate for where the inter-cluster transition should occur.

The edge cost, or rather weight, for a neighbor pair is referred to as the neighbor transition cost, $J(Ne(A,B))$. As the neighbor transition cost is the minimal cost to transition from one cluster to another by definition, the neighbor transition cost is used within the neighbor based method as the cluster to cluster cost. This obviously can produce a very different clustering than the centroid based method but both are equally valid approaches. (In the directional edge case, where the neighbor transition cost to go from cluster A to cluster B is not the same in the reverse direction, the average of the neighbor transition costs may be used.) Creation of the neighbor pairs for each pair of clusters adds an additional $O((k_\alpha^2)(p_{c,max})^2)$ to compare the cost of every member in every cluster to every member of every other cluster. However, even at the bottom level of clusters k_α is as small as $1/p_{c,max}$ time the size of the target set T. Similarly, in cases when k_α is larger than this minimum, the average number of members per clusters is less than $p_{c,max,}$ and hence

The algorithm for the neighbor pair creation is straightforward but is described in Pseudo-Code 5.1 below. In the function, GetBestNeighborPair(), for each member of the first cluster, j, the algorithm looks through all members of the second cluster, k, and then selects the pair with the lowest cost. During this process, the second best edge starting from the neighbor pair member within cluster j into the opposite cluster, k, is also stored. This second best edge is referred to as the neighbor "back-up" edge, $Ne_b(K_{\alpha,j}, K_{\alpha,k})$ and the back-up edge for both neighbor pair members are also shown in Figure 5.4 as dashed-lines. Storing this back-up edge information will be shown to be necessary if the neighbor information is used in solving the cluster sub-problems.

```
1. for j < k_α
2.     for k < k_α
3.         if (K_α,j != K_α,k)
4.             {Ne(K_α,j, K_α,k), Ne_b(K_α,j, K_α,k)}= GetBestNeighborPair(K_α,j, K_α,k)
5.             {Ne(K_α,k, K_α,j), Ne_b(K_α,k, K_α,j)}= GetBestNeighborPair(K_α,k, K_α,j)
```

Pseudo-Code 5.1: Neighbor Pair Creation

5.5.4 Neighbor Based Cluster Sub-Problems and Neighbor/Centroid Mixed Cluster Sub-Problems:

As the neighbor pair defines the lowest cost transition points between clusters, this information can also be used to more cleverly setup the sub-problem TSPs to obtain a better estimate of the cluster cost and resource usage. However, in order to take full advantage of the neighbor pair information this method must now consider cluster sub-problems in terms of triplets of clusters, i.e. there is the cluster that is being solved as a TSP sub-problem but there is also the cluster that the trip could be potentially coming from and the cluster that the trip is potentially going to, which influence required starting and ending targets for the sub-problem TSP. This is demonstrated in Figure 5.5.

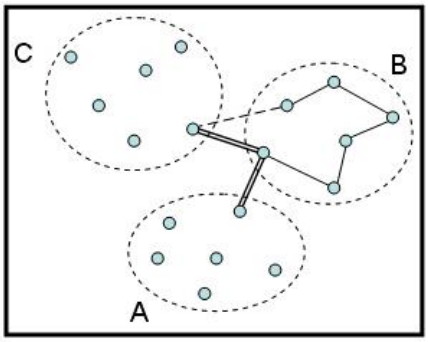

Figure 5.5: Considering sub-problems as cluster triplets using neighbor pair information. Clusters' neighbor pair members are connected with a doubled line; a cluster back-up edge is shown as a dashed line; solid lines within cluster B are the solution to the TSP sub-problem of cluster B.

In the process of determining cost and resource estimates for cluster B, the neighbor based sub-problem method will also determine estimates for the possibility that a potential trip is coming "into" cluster B from all other clusters A, where A !=B, and going "onto" all clusters C, where C != (A || B). The number of TSPs for cluster level α then becomes for the general case $k_\alpha(k_\alpha\text{-}1)(k_\alpha\text{-}2+1)$ for starting at one cluster going to another cluster and then going onto a third clusters, plus $s(k_\alpha)(k_\alpha\text{-}1+1)$ for the case of starting at a source instead of a cluster. The "plus one" at the end of each equation is included for the case where the trip doesn't go onto another cluster but rather ends at that cluster.

This is a very significant increase in the number of TSPs that need to be solved. In order to reduce this number a combination of the centroid and neighbor based sub-problem was investigated. In this combined approach, as shown in Figure 5.6 where it is assumed that the method is currently determining cost and resource estimates for cluster B, the method only needed to consider cluster B to all clusters C

where C != B. This reduction is accomplished by solving all cluster B TSPs starting at the cluster B's centroid but ending at a member within cluster B that has the lowest weight edge to the centroid of cluster C. This reduces the number of TSPs to $k_\alpha(k_\alpha-1+1)$ where the plus one is again included for the case where the trip doesn't go onto another cluster but rather ends at that cluster. However the estimates produced by this method may not be as good as those produced from the sole neighbor based sub-problem method but they still have the potential to be better than the sole centroid based sub-problem method.

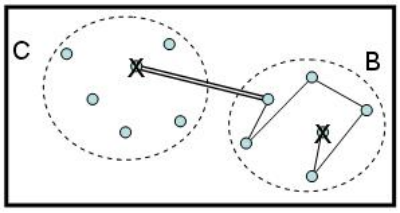

Figure 5.6: Considering sub-problems as cluster triplets using centroid-neighbor pair information. Clusters' centroids are labeled with an x; clusters' centroid neighbor pair members are connected with a dotted line; solid lines within cluster B are the solution to the TSP sub-problem of cluster B.

In both neighbor based cluster sub-problems it may be the case that the best member to enter the investigated cluster at and the best member to leave the investigated cluster may be the same cluster as shown in Figure 5.5. In this special case, the back-up neighbor edge associated with the leaving member is used in the TSP solution for the sub-problem estimate instead.

5.5.5 Analysis of the Neighbor Based Cluster Sub-Problem and Neighbor/Centroid Combined Cluster Sub-Problem Methods

Despite the better estimates of the solely neighbor based sub-problem method and the combined neighbor/centroid sub-problem method, in both cases it was found that small improvements in the solution to the overall task allocation problem were obtained but the required additional computation time was too significant for these approaches to be of value. Overall, it was found that the sole neighbor based sub-problem method produced final answers for the test problems discussed in detail in Section 5.10 that were up to 3-4 percent error better than any of the methods investigated. However, as the analysis of Section 5.5.4 of the number of TSPs required suggests, not only were the computation times worse, and in some cases more than orders of magnitude worse than any other method, the sole neighbor based sub-problem method scaled very poorly as well.

The combined method did run significantly faster than the sole neighbor based sub-problem method but it did not show any significant improvement over using the centroid based sub-problem estimates in the overall percent errors that are reported in Section 5.11. For these reasons, this paper will focus only on the cases where the cluster to cluster costs are calculated via either the centroid based or neighbor based methods and the cluster sub-problem estimates are calculated via the centroid based methods as together these methods showed the greatest benefit in terms of either or both computation time and percent error as will be discussed in Section 5.11.

5.5.6 Further Time Reductions for a Resource Special Case

Under a special constraint condition when using the centroid based sub-problem method, it is possible to not have to solve any TSPs. This condition states that

constraints must only be resource usage based and the resource usage is only significant at the targets themselves, i.e. the resource use in transitioning between states is negligible. An example of this might be in a delivery problem when the fuel is not a concern but perhaps the storage space in the transport vehicle is. In cases like these, the resource usage for a cluster may be estimated as simply the sum of the cluster's member's resource usage.

The reason that this sub-problem solving method substitution can occur is because in the centroid based sub-problem the cost estimates are not necessarily needed. This may seem counter intuitive but can become clear by first recognizing that the centroid sub-problem solution method works independently of the method chosen in the clustering costs and visa versa. Therefore the clusters will also be formed independent of the centroid sub-problem solution. Since all targets and hence all clusters must be included in any valid solution to the problem, the sum of all cluster cost estimates at any level will always be the same regardless of what trips are decided in the summary problem or other level of the algorithm. Therefore, the later methods discussed in this paper would only have to be concerned with the transition costs between clusters and not the costs estimates internal to the clusters. Furthermore, since in this case the resource estimate is obtainable through other methods, the overall H-G^*_{TA} method is no longer required to solve the TSPs and hence the entire H-G^*_{TA} can be improved significantly.

Despite the obvious benefit in removing the need to solve TSPs, the results presented in Section 5.11 of this paper solve the TSP in all centroid based sub-problem variation as a means for calculating the resource/constraints estimate. This also serves the focus of this paper to present the performance of the H-G^*_{TA} in the most general

case and conservative manner. Regardless, for the centroid based recursion case that will be presented in Section 5.7.1, these TSPs sub-problems are also necessary for "backing out" the final trip of targets from the solution to the summary problem.

5.6 Solving the Top Level Summary Problem

At the top level of the cluster hierarchy, where the number of top level clusters, k_{top}, equals exactly $k_{top,max}$, the H-G$^*_{TA}$ method now considers the source input set, S, as well. The first step in the H-G$^*_{TA}$ method is to relate the clusters created in the earlier stages to the sources. This is done by either the centroid based or the neighbor based cluster transition cost method where the source is simply treated as a single member cluster. It is important to note, however, in order for the neighbor based cluster transition cost method to produce top level clusters that can be related to the sources, these neighbor based transition costs from the source to the clusters must be set for clusters of every level of the cluster hierarchy. Otherwise, the source to cluster member costs at the top level would be undefined.

Once the source to cluster transition costs have been established, and since the cluster to cluster transition costs were established in the previous stage, this allows the H-G$^*_{TA}$ method to treat the top level clusters as targets and solve an MTSP of s sources and k_{top} targets using the already proven G$^*_{TA}$ method [5.18]. As the MTSP of the summary problem has been controlled to be of a reasonable size by the input $k_{top,max}$ in the previous stages, the runtime for this stage is also controlled to be within an acceptable limit.

The G$^*_{TA}$ method is a well established optimal MTSP solution method that has been shown to run particularly fast especially when compared to traditional MILP

implementations [5.18],[5.30]. However, even when using a fast MTSP method like G^*_{TA}, solving for an optimal solution greatly restricts the size MTSP that can be solved in a real-time application. Furthermore, as mentioned with regards to a previous section, it is generally desirable to have $k_{top,max}$ be as large as possible since the summary problem will then more closely resemble the original problem. Therefore in order to solve a summary problem with a significantly greater $k_{top,max}$ the guarantee for optimality at the summary problem level can be dropped as this could allow the use of faster approximation methods and the H-G^*_{TA} is designed as an approximation method.

Several approximation methods were attempted, including the greedy method described in Section 5.9, as a method for solving the summary problem within a range of $k_{top,max}$ problems that varied in size from $k_{top,max} = 6$ up to $k_{top,max} = 30$ at which point the approximation method runtimes became too large to meet the overall goals of this paper's research. All of these methods however resulted in relative percent errors that were on average around 20-30% higher than those found using the G^*_{TA} method with the smaller $k_{top,max}$ values.

Although it may be possible that there does exist an approximation method that could produce reasonably close if not better end percent errors with larger $k_{top,max}$ values, none were found in this study. This does however help to demonstrate the value of having a high accuracy, or in this case optimal, solution to the summary problem. Furthermore this also provides a higher level of confidence in the clustering stages ability to summarize the original large problem as otherwise it is unlikely that there would have been as large of a difference in using an approximation or an optimal method to solve the summary problem.

5.7 Determining the Final Solution through the Recursive Breakdown of the Summary Problem Solution

Although the solution to the summary problem assigns the top level clusters to the sources, and hence which subset of targets is assigned to each source, it does not directly define the complete order in which the original targets are assigned. In order to determine each source's final trip, the assigned top level clusters must be recursively broken down using one of the two new methods contributed by this section of the paper.

5.7.1 Recursive Centroid Based Final Trip Determination

The first of the two methods again follows the more traditional design of placing a strong emphasis on the cluster centroid and hence will be referred to as centroid based recursion. This method also requires the use of the TSP solutions generated from solving the earlier sub-problems using the centroid based sub-problem method. This does not however require that the centroid based transition costs be used in the clustering stage but only that the centroid based TSP solutions are available before the start of this centroid based recursion, regardless of how the sub-problems were solved prior to the summary problem stage.

The inputs to the centroid based recursion method are the clusters, the trips defined as a part of the summary solution, $Tr_{i,top}$ for all $i \in S$, and the TSP solutions obtained through the centroid based sub-problem method. Using these inputs, the centroid based recursion method follows Pseudo-Code 5.2 and 5.3 written below.

getTripElement(q,Tr)	Return the q^{th} element of trip, Tr. If q is not within the length of Tr, return NULL

```
1.  for all source vertex v_i , where i ∈ S
2.      create trip Tr_i,final containing only source vertex v_i
3.      for q = 1 to tripLength(Tr_i,top)
4.          K_top,q  = getTripElement(q,Tr_i,top);
5.          Tr_TSP = getTSP(K_top,q)
6.          recursiveTripMaker_Centroid(K_top,q, Tr_TSP, Tr_i,final)
```

Pseudo-Code 5.2: The Highest Level of Centroid Based Recursion

The centroid based recursion method starts by creating a trip, $Tr_{i,final}$ for all $i \in S$, to store each source's final trip in Pseudo-Code 5.2, line 2 and then in Pseudo-Code 5.2, line 3, it begins examining each source's summary solution trip, $Tr_{i,top}$, individually. For each source's summary trip, $Tr_{i,top}$, the method investigates each top level cluster, $K_{top,q}$, in that trip in the order specified by that trip. Then each top level cluster, $K_{top,q}$, is sent to the function recursiveTripMaker_Centroid along with Tr_{TSP}, the TSP solution for that top level cluster that was obtained earlier using the centroid based sub-problem method. The source's final trip, $Tr_{i,final}$, is also sent to the recursiveTripMaker_Centroid function in order to store the results of the recursion. The recursiveTripMaker_Centroid function is then run according to the pseudo-code written below where the line numbering is continued from above only to allow both sections of code to be discussed collectively.

```
7.  recursiveTripMaker_Centroid (C_input, Tr_input, Tr_final){
8.      if (the members of C_input are also clusters)
9.          for q = 1 to tripLength(Tr_input)
10.             C_q      = getTripElement(q,Tr_input);
11.             Tr_TSP = getTSP(C_q)
12.             recursiveTripMaker_Centroid(C_q, Tr_TSP, Tr_final)
13      else if (the members of C_input are targets)
14.         for q = 1 to tripLength(Tr_input)
15.             v_q = getTripElement(q,Tr_input);
16.             add2Trip(v_q, Tr_final)
17. }
```

Pseudo-Code 5.3: Centroid Based Recursion, Recursive Function Definition

In short, the recursiveTripMaker_Centroid function works in two parts, which are separated according to the if statement of Pseudo-Code 5.3, line 8. If the members of the input cluster, C_{input}, are also clusters, the recursion continues to go deeper investigating each one of C_{input}'s cluster members in the order specified by Tr_{input}, where Tr_{input} is the TSP solution to C_{input}'s sub-problem. If the members of the input cluster, C_{input}, are targets however, the recursion bottoms out in Pseudo-Code 5.3, lines 13-16 and the targets are added to the final trip in the order decided upon in the solution to the C_{input} sub-problem, i.e. in the order of Tr_{input}.

This process becomes clearer upon reviewing Figure 5.7 which steps through the process for a problem having three levels of clusters. Figure 5.7v, shows the top level of the summary solution for an arbitrary source where only $C_{top,1}$, the first cluster of the top solution trip is shown. Since the source is assigned $C_{top,1}$ the source is also assigned all of the sub-clusters members of $C_{top,1}$ and all of the members of the sub-

cluster's members and so forth. Figure 5.7a, shows the cluster $C_{top,1}$'s members in an arbitrary order to demonstrate the generality of the example. Figure 5.7c then shows cluster $C_{top,1}$'s members in the order according to $C_{top,1}$'s centroid based sub-problem TSP solution as shown in Figure 5.7w. This is same order that $C_{top,1}$'s members are then sent to the recursive function call in Pseudo-Code 5.3, line 12.

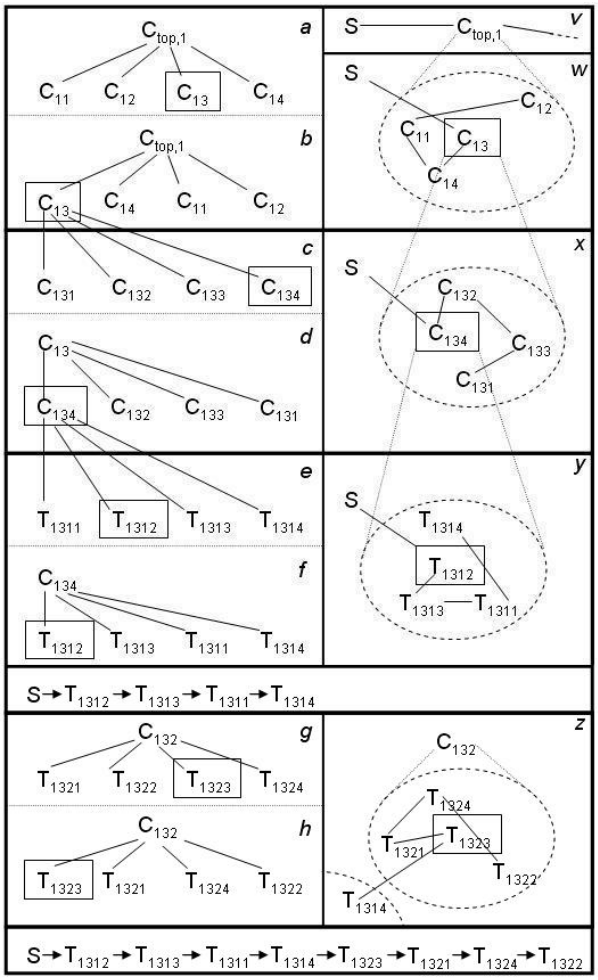

Figure 5.7: Partial example of centroid based recursion applied to partial trip top level trip shown in v. a-h) tree representation of the cluster hierarchy where the cluster members are organized arbitrarily in a,c,e and g and in b,d,f,h they are ordered according the cluster's centroid based sub-problem TSP solutions shown in w,x,y,z respectively. Some elements are boxed to signify these elements as the centroid for that cluster.

Figure 5.7c then shows the members of C_{13}, the first element in $C_{top,1}$'s TSP solution, in an arbitrary order. Then Figure 5.7d shows again the members of C_{13}, but

specified in the order of its sub-problem TSP solution as shown in Figure 5.7x. This process continues until finally at the bottom level, the targets shown are added to the source's final trip in the order specified by the bottom level cluster's sub-problem TSP. Once this branch of the recursion has bottomed out, the method returns to one level above in the recursion, C_{132}, and the process repeats itself as shown in Figure 5.7g, 5.7h and 5.7z. Here the recursion bottoms out again, adding the members of C_{132} to the source's final trip in the order specified by the C_{132} centroid based sub-problem TSP solution, and the recursion method continues to follow the same pattern from there.

The solutions generated by the centroid based recursion have a general form represented in Figure 5.8 where the partitions of the bottom level clusters are shown as dashed line circles. Figure 5.8 demonstrates how the order that the lower level clusters are visited in is determined by the trip of the higher level. As can be seen, the source moves into every cluster by starting at centroid and then ending at the target that was best for that cluster's TSP sub-problem. The source then moves onto the next cluster's centro0id without any concern of where it ended its last trip in the previous cluster.

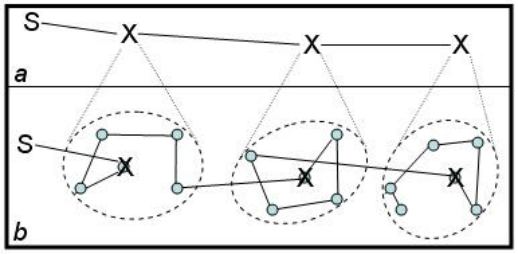

Figure 5.8: Determining the source S's final trip using centroid based recursion. Both the bottom level of clusters and the solution at the above level are shown in b and a respectively. Grey dots represent targets and x's represent the clusters at the above level and hence are also the near-centroids of bottom level.

As is clear by Figure 5.8, this does not always make for the most locally efficient solutions. However, this recursion method is very fast at obtaining a solution to the original problem especially if the centroid based TSP sub-problem solutions were already calculated in the clustering stage. If the centroid based TSP sub-problems were already solved, then the cluster based recursion method does not require solving any additional MTSP or TSP problems. Thereby, the cluster based recursion method only needs to step through all clusters once via the trips that they are a part of, whether those be trips from the summary problem solution or from a centroid based TSP sub-problem solution. This makes the cluster based recursion method O($|K|$) plus O($|T|$) for the stepping through the TSPs of the bottom level clusters which have targets as members.

In return for this speed, the cluster based recursion method relies heavily upon post-optimization methods for improving the final solution's accuracy, such as the methods discussed in Section 5.8.

5.7.2 Recursive Neighbor Based Final Trip Determination

The characterization of clusters through the neighbor pairs defined in Section 5.5 can also be used to aid in the recursive reassembly of the targets from the summary solution and hence this procedure will be referred to as the neighbor based recursion method. This method takes all inputs that were specified in the task allocation problem definition of Section 5.2, the cluster set K, the summary solution trips, $Tr_{i,top}$ for all $i \in S$, and the neighbor pairs for all clusters at the same level of the cluster hierarchy as well as the source/cluster neighbor pairs.

The neighbor information is crucial as this method attempts to find higher accuracy final solutions by resolving the TSP sub-problem for each given cluster depending upon where the summary solution trip originates from before that given cluster and where the summary solution trip will be going after that given cluster. This method is described in Pseudo-Code 5.4 and 5.5 below and then each case within the pseudo-code is then explained using Figures 5.9 and 5.10.

getTripElement(q,Tr)	Return the q^{th} element of trip, Tr. If q is not within the length of Tr, return NULL

```
1.  for all source vertex v_i , where i ∈ S
2.      create trip Tr_i,final containing only source vertex v_i
3.      v_i,copy = copyState(v_i)
4.      for q = 1 to tripLength(Tr_i,top)
5.          K_top,q   = getTripElement(q,Tr_i,top);
6.          K_top,q+1 = getTripElement(q+1,Tr_input);
7.          recursiveTripMaker_Neighbors(v_i,copy, K_top,q, K_top,q+1, Tr_i,final)
```

Pseudo-Code 5.4: Highest Level of Neighbor Based Recursion

As can be seen, the Pseudo-Code 5.4 is almost identical to the centroid based recursion Pseudo-Code 5.2. The main difference however is the call to recursiveTripMaker_Neighbors which also takes the next cluster in the source's summary solution trip, $K_{top,q+1}$, and a copy of the source vertex data, $v_{i,copy}$, whose state will be allowed to be altered within the recursion method. Notice also that the TSP solution from the sub-problem is not necessary as the neighbor based recursion will be resolving the TSPs regardless of the sub-problem solutions. The pseudo-code for recursiveTripMaker_Neighbors is shown below:

```
8.  recursiveTripMaker_Neighbors(v_source, C_input, C_next, Tr_final){
9.      if (C_input is a cluster and not a target)
10.             if (C_next != NULL)
11.                     v_end = member of C_input ∈ Ne(C_input, C_next)
12.             else
13.                     v_end = NULL
14.             Tr_TSP = G*_TA(v_source, members of C_input, v_end)
15.                     for q = 1 to tripLength(Tr_TSP)
16.                             C_q     = getTripElement(q,Tr_input);
17.                             C_q+1   = getTripElement(q+1,Tr_input);
18.                             if ((C_q+1 == NULL) && (C_next != NULL))
19.                                     C_q+1 =  member of C_next ∈ Ne(C_input, C_next)
20.                             recursiveTripMaker_ Neighbors (v_source, C_q, C_q+1, Tr_final)
21.     else if (C_input is a target and not a cluster)
22.             add2Trip(C_input, Tr_final)
23.             v_source = copyState(C_input)
24. }
```

Pseudo-Code 5.5: Neighbor Based Recursion, Recursive Function Definition

The function recursiveTripMaker_Neighbors takes in as inputs v_{source}, the state of the source in the recursion, C_{input}, which is either a cluster or a target vertex, C_{next}

which is the cluster or target vertex immediately following C_{input} in the trip that was being inspected when recursiveTripMaker_Neighbors was called, and Tr_{final} which is the trip used to store the final trip for the currently considered source. If C_{input} is not a target but rather a cluster, the neighbor based recursion method will resolve a TSP of the C_{input}'s members taking into consideration where the trip from which C_{input} came is going to next, i.e. C_{next}. If C_{next} is not NULL, then the member of C_{input} that is part of the neighbor pair of C_{input} and C_{next} is stored as v_{end}.

This last case is better explained with the aid of Figure 5.9 which shows the neighbor based recursion being applied to a summary solution trip at the second to bottom layer of a multi-layer cluster hierarchy problem. Figure 5.9a is the second to bottom layer trip solution for the shown source, i.e. S $\rightarrow C_1 \rightarrow C_2 \rightarrow C_3$. In order to demonstrate how these clusters' TSP are re-solved, the clusters' neighbor pairs are shown in Figure 5.9b as double lined edges. Recognizing from the top layer trip that the source will have to start at its initial state and "travel" through C_1 onto C_2, the member of C_1 that is part of the (C_1, C_2) neighbor pair is then selected as v_{end} by Pseudo-Code 5.5 line 11, where C_1 is C_{input} and C_2 is C_{next}. With v_{end} established, a TSP is solved using the members of cluster C_{input} as targets, v_{source} as the source state, and v_{end} as the required end target of the TSP as is shown in Figure 5.9c.

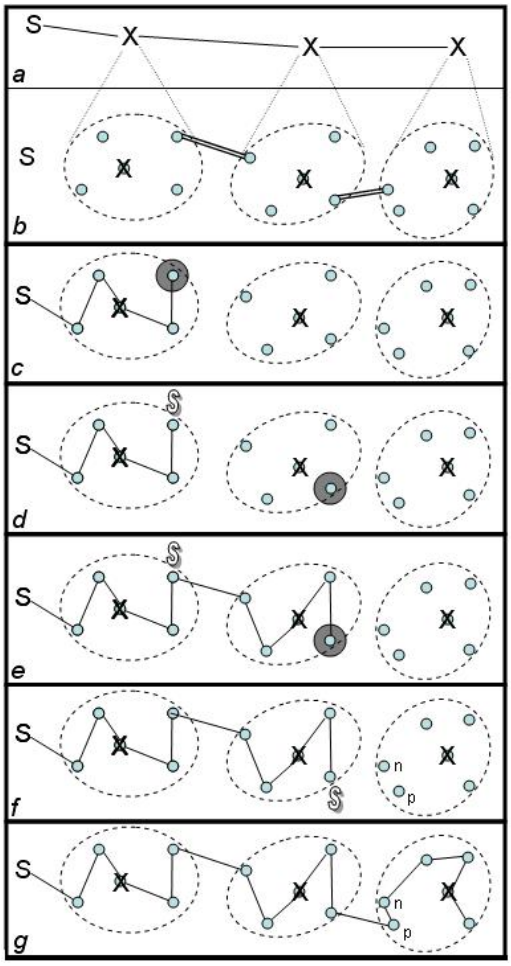

Figure 5.9: Determining the source S's final trip using neighbor based recursion. light grey dots are targets, dark grey dots are required end targets and x's are the clusters at the above level and hence are also the near-centroids of bottom level a) the above level solution b-g) resolving sub-problem TSPs for bottom level c-e) the white S represents the starting point for re-solving the next clusters' TSP sub-problem

The G^*_{TA} method was chosen to solve this TSP since as described in [3.18] and [3.30] it has been shown to solve these kind of problems optimally and quickly. In addition, as is discussed in [5.30], it can also handle both conditions of having

required end targets, i.e. the source(s)' trips must end at a pre-specified target, or not having any end targets, i.e. the source(s)' trip may end at any target.

As is shown in Figure 5.9d, the purpose of setting v_{end} is to allow the end of the C_{input} TSP trip that is being created to end at a strategic position for moving onto C_{next}. Hence with the C_1 TSP stored in the final trip, Tr_{final}, via Pseudo-Code 5.5 line 22, at the end of this TSP, Pseudo-Code 5.5 line 23 sets v_{source} state as the last target's state. This is signified by the white "S" in Figure 5.9d.

The recursion then returns to the for loop of Pseudo-Code 5.4 lines 4-7 where in the next call to recursiveTripMaker_ Neighbors C_2 is C_{input} and C_3 is C_{next}. In Pseudo-Code 5.5 line 11, v_{end} is set once again, and with v_{source} set to the new state from before, i.e. at the white "S" in Figure 5.9d, this TSP now can be solved optimally for the best way to move through the cluster C_{input} under the strategic assumption of v_{end} as shown in Figure 5.9e.

The process then continues in the same fashion both adding the targets to Tr_{final} from the C_2 resolved TSP and in setting v_{source} as the end of that TSP. In the next iteration however as C_3 is the end of the high level trip, when C_3 is sent as C_{input} to recursiveTripMaker_ Neighbors, C_{next} is set to NULL. In the case where C_{next} is NULL, v_{end} is also NULL in line 13 and no restrictions are placed where the corresponding TSP trip ends as is shown in Figure 5.9f and Figure 5.9g.

It can also be seen that since the neighbor based method only fixes the endpoint, within any mid-trip clusters such as C_1 and C_2, v_{source} is always at a state outside of the cluster that is being investigated. Examining Figure 5.9f and Figure 5.9g

closely it can be seen that although the target labeled "n" is part of the (C_2,C_3) neighbor pair, having the v_{source} be external to C_3 allows entry into that cluster to be at any target that will lead to the best TSP regardless of the neighbor pair, which in this case is at target "p".

There is one additional case to discuss that can only be seen in higher levels of the cluster hierarchy and it is demonstrated within Figure 5.10. The clusters and targets are named in Figure 5.10 to indicate their relationship in the hierarchy and, without any loss of generality, they are also drawn in the same order they would appear in the resulting TSP trips from applying neighbor based recursion's TSP re-solving. For example, the source's top level trip of the Figure 5.10 is $S \rightarrow C_1 \rightarrow C_2$ and the resolving of C_1's sub-problem TSP would result in $S \rightarrow C_{11} \rightarrow C_{12}$ and likewise the resolving of C_{11}'s sub-problem TSP would result in $S \rightarrow T_{111} \rightarrow T_{112}$. Furthermore, and again without any loss of generality, the clusters and targets are also arranged so that cluster members on the same level who are next to each other but are in different clusters are the neighbor pair members of their parent clusters, i.e. T_{112} and T_{121} are the neighbor pair members of *Ne*(C_{11},C_{12}). The clusters and targets are named and drawn in this fashion to allow Figure 5.10 and the discussion of the additional case to be easier to follow.

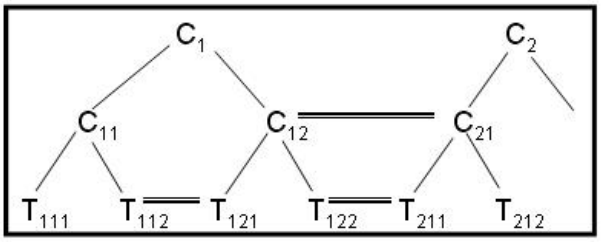

Figure 5.10: 2 level cluster hierarchy, where only a top level summary solution trip is shown along with the complete C_1 hierarchy as well as part of the C_2 hierarchy to demonstrate neighbor relationships. Single lines indicate hierarchical relationships. Dash-Dot lines indicate the top level summary solution trip. Double lines indicate the neighbor pair members of the clusters that those sub-clusters or targets belong to.

Returning now to Pseudo-Code 5.5 line 18, in the case when C_q is at the end of a trip, C_{q+1} is NULL at this line. If C_{next} is also NULL, C_{q+1} will remain NULL allowing the next recursion call's TSP to have no restrictions on where it ends (i.e. v_{end} for the next recursion call will be NULL). If C_{next} is not NULL however, the algorithm can take advantage of the C_{input}, C_{next} neighbor pair information to still set a value for C_{q+1}. This later case is shown in Figure 5.10 when C_q is C_{12}, and therefore as C_{12} is then end of the C_1 sub-problem's TSP trip and C_{q+1} is NULL in Pseudo-Code 5.5 line 18. In addition, since this also implies that C_{input} is C_1, C_{next} is C_2.

Furthermore since C_q is at the end of C_{input}'s trip, this means that it is a part of the (C_{input}, C_{next}) neighbor pair, as C_q in this case would have been declared as v_{end} for the C_{input} sub-problem TSP resolving earlier in Pseudo-Code 5.5 line 14. Therefore, even though C_{input}'s trip has ended, the other half of the neighbor pair of C_{input} and C_{next}, which is in C_{next}, is a likely cluster (or target) that can be considered as C_{q+1}. Following the Figure 5.10 example, this means that in following recursion call on line 20, when C_q is C_{12}, C_{q+1} is the neighbor pair member of C_2, which is C_{21}.

This last case only occurs in the Pseudo-Code 5.5 when line 18 is true and then Pseudo-Code 5.5 line 19 is executed. These lines are actually very important as this is what allows the neighbor based recursion method to intelligently move from one cluster to another even if those clusters are sub-clusters within different top level clusters as was described in with Figure 5.10.

5.8 Optional Post-Optimization

After recursively breaking down the summary solution into trips for the sources that contain only the targets, i.e. members of T, the H-G^*_{TA} method may exit with these trips as the final solution. However, as was mentioned specifically in the centroid based recursion description, Section 5.7.1, a post-optimization technique can be applied to attempt to improve these trips. This section discusses the incorporation of the K-Opt method [5.31] as a fast and effective post-optimization technique.

The K-Opt method is a well established method that is very effective at reviewing and improving upon any problem whose solution can be represented as a linear chain or sequence of length, L_{seq}, where the order within that chain can be easily altered without adversely affecting the solution's validity. The K-Opt method can be easily explained with the aid of Figure 5.11. The K-Opt method begins by examining the first *kopt*-elements, as shown in the in the Figure 5.11a example, by considering every possible ordering of those *kopt*-elements while keeping the rest of the chain unchanged, as shown in Figure 5.11b. If one of the new arrangements of those *kopt*-elements results in an overall solution that is better than the original, for example the new solution has a lower cost as shown in Figure 5.11b, the originally investigated *kopt*-elements are re-arranged to match the best newly found sub-sequence

arrangement. This rearrangement is shown in Figure 5.11c. The K-Opt method then repeats this process for the next set of *kopt*-elements, starting at only one element further down the sequence chain than it did in the previous examination, as shown in Figure 5.11d.

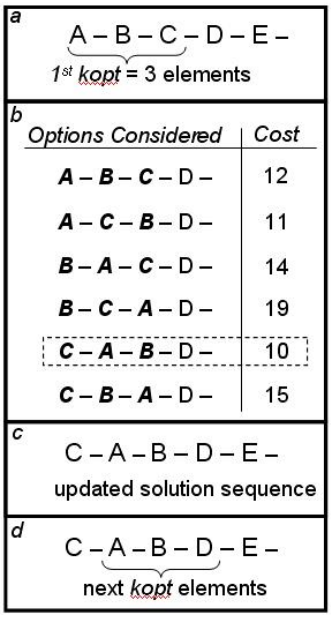

Figure 5.11: An example of K-Opt being applied to a solution sequence where *kopt* = 3. a) Segment of the solution sequence before applying K-Opt to the B-C-D sub-sequence b) K-Opt generated options from examining the B-C-D sub-sequence c) Updating the original solution sequence with the lowest cost option from b) d) Advancing of the K-opt algorithm to the next *kopt* sub-sequence.

This process is continued until it reaches the $(L_{seq} - kopt)^{th}$ element of the sequence chain as past this point the sequence is too short to consider *kopt* elements at once. However, this method may also be run iteratively over the same sequence chain several times where the improvement for every K-Opt run down the sequence chain is

recorded and the iterations are stopped once the improvement drops below a threshold or equals zero.

In general the larger the value of *kopt* in the K-Opt method, the better the ending solution but the longer the amount of time the method will take. In addition, the big "O" runtime of a single run of K-Opt down a sequence chain is shown in (5.8). Consequently, the iterative method obviously can take considerably longer than a single run application of K-Opt, particularly if the initial solution inputted into K-Opt is rather poor.

$$O\left(\left(L_{seq} - kopt\right)\left(_{kopt}P_{kopt}\right)\right) \tag{5.8}$$

The key then becomes to apply K-Opt in situations where the chain is more likely to only have small sections of imperfection in between significant length sections that are already fairly good if not optimal. This way the *kopt* of K-Opt can be rather small to essentially "untangle" the small sections of imperfection and as the larger sections are difficult if not impossible to improve upon the iterations should end relatively quickly.

For these reasons, K-Opt was selected as the post-optimization technique for the H-G^*_{TA} method. With either recursion method, the largest source of easily identifiable error in the recursion solution's trips will most likely be in the transition between clusters. This is clearly seen in the centroid based recursion as the end of each TSP within each cluster is independent of where the trip is heading next and likewise the beginning of each cluster's TSP is also set to be that cluster's centroid independent

of where the trip was previously. Similarly, in the neighbor based recursion, although the previous and next elements within a trip are considered in the clusters' TSPs through the use of the neighbor pairs, the fixed end vertex of the TSPs is only a strategic estimate of which cluster member may be the best exiting member. Therefore the K-Opt post-optimization may be able to help in the centroid based recursion as well.

As K-Opt is incorporated into the H-G^*_{TA} to examine mostly these cluster transitions, the *kopt* of K-Opt was set to 4, and will be referred to as 4-Opt for this specific case. Given the big "O" runtime of equation (5.8), the runtime for 4-Opt is reasonable. Experimentally 4-Opt's runtime was only slightly greater than running K-Opt with *kopt* set to 3 (an increase of up to approximately 3 ms on average under the conditions of the experiments in Section 5.10) but still improved the neighbor based recursion by up to 3 percent error. Running K-Opt with *kopt* set to 5 took significantly longer without showing significant improvement, particularly in the neighbor based recursion. Hence, the experiments presented in Section 5.11 of this paper are limited to the 4-Opt case.

Larger *kopt* values may still lead to further improvement, however, but given that the neighbor based recursion provides locally optimal solutions (to within the intelligent transition estimates provided by the neighbor pairs) in order to observe a significant improvement in general over the recursion solution's trips, *kopt* of K-Opt would have to be increased to a value greater than twice that of $p_{c,max}$, i.e. the size of the clusters. This would allow the K-Opt method to essentially be able to change the bottom level cluster order as well. However, this would cause *kopt* of K-Opt to be

significantly large and therefore in this case, running K-Opt would have too large of a negative impact on the computational runtime.

For specific applications, particularly those whose solutions have well understood characteristics that could be exploited, other post-optimization techniques can be applied instead of or in addition to K-Opt without affecting the rest of the H-G^*_{TA} method.

5.9 The Greedy Comparison Method

The standard approximation method used for comparison is a greedy approximation algorithm well known for its speed. Although not usually highlighted in its discussion, greedy approximation methods for MTSPs are typically able to provide notably respectable approximations of the optimal solution for random problems, specifically for problems similar to those that will be considered in the results of Section 5.11.

When comparing greedy MTSP methods to other MTSP approximation methods, in general, greedy based MTSP methods are very similar in design and hence performance to many of the instantaneous assignment market-based approximations that are discussed in [5.24]. In practice, however greedy methods are often chosen over instantaneous assignment market-based approximations and other methods that can offer 2-approxiations, often for their simplicity and solution quality [5.6],[5.24]. Furthermore, while the greedy method presented here is a centralized algorithm, as is the H-G^*_{TA} method, many market-based and other approximation methods focus on decentralized approaches and have traditionally not been implemented on problems of this study's scale to the best of the authors' knowledge. Therefore, in addition to the greedy method's satisfactory solution quality, reputation for speed, practicality for the

scale of problems investigated, and its centralized nature, this greedy method is also a standard that is well understood and hence makes for a creditable basis of comparison.

To provide insight into the using a greedy method as a standard of comparison, it is important to recognize the method's faults as well. It is well established, that there are no known greedy methods that can guarantee to optimally solve a TSP or MTSP problem. Furthermore, it is not difficult to find special cases where greedy methods can do very poorly since a series of closely spaced points can lead a source "astray" from what might be the optimal trip as is shown in Figure 5.12a. These cases lead greedy methods to commonly have a 2-approximation bounds as has been shown for several of the methods described in [5.6].

Intuitively, it can be seen however, that these faults can be partially overcome, particularly in MTSP problems where $|S|$ is not small. Figure 5.12a-f illustrates this point where starting with a TSP in Figure 5.12a, each consecutive panel set adds additional sources and as a result the greedy solution more closely resembles the corresponding optimal solution. With more sources spread throughout the space, the more likely it is that even if one source is led "astray" there is another source that may be able to help overcome this problem. Therefore, in random environments, especially with more sources available, the better the chance that on average that a greedy method may perform well.

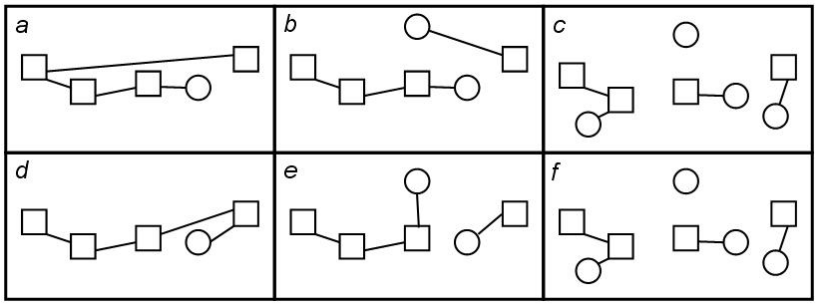

Figure 5.12: Greedy vs. Optimal Solutions for TSP and MTSP problems. As the number of sources increases so does the performance of greedy method in general. a-c) Greedy solutions d-f) respective optimal solutions

The greedy method used in this study for comparison is the same greedy method that was used in [5.18],[5.30] as this greedy method was shown to be very fast and at the same time be accurate enough to provide effective upper bounds for finding the optimal solutions sought in [5.18],[5.30]. The upper bounds in [5.30] were effective enough to reduce the computation time on average by more than 70% than running the optimal G^*_{TA} method without the greedy upper bound component (referred to as A^*_{TA} in [5.30]).

For completeness, this greedy method is shown first in Pseudo-Code 5.6 and then described briefly below. Details pertinent to the overall runtime of the method are also highlighted here so that they may be discussed in a later comparison with the H-G^*_{TA} method in Section 5.11.

SortEdges[v_k]	sort all outgoing edges of vertex, v_k
AddTarget[v_j,Tr_i]	add vertex v_j to the end of trip, Tr_i and remove it from U
[v_j,Tr_i]=GetSmallestEdge	returns Tr_i the trip with the smallest edge out of its last vertex to an unassigned target v_j.
ConstraintCheck[v_j,Tr_i]	if the constraints check for adding target v_j to trip Tr_i, return true
Remove[v_j,Tr_i]	remove from consideration the edge from Tr_i 's end to vertex v_j
ValidEdge[Tr]	returns true if there is an edge from the end of any trip, Tr, to an unassigned target.

```
1.  U = { all unassigned targets }
2.  for all vertex v_k ∈ V
3.        SortEdges[v_k]
4.  for all source vertex v_i ∈ S
5.        Tr_i = { v_i }
6.  while (U != {NULL} && ValidEdge[Tr])
7.          [v_j,Tr_i] = GetSmallestEdge
8.          if ( ConstraintCheck[v_j,Tr_i] )
9.                  AddTarget[v_j,Tr_i]
10.         else
11.                 Remove[v_j,Tr_i]
12. if (U != {NULL} ):  return solution cost as Ĵ
13. else:               return infinity
```

Pseudo-Code 5.6: The G_{TA} Greedy Task Allocation Approximation Method

In Pseudo-Code 5.6 lines 2-3, the G_{TA} algorithm begins by creating a separate sorted list of the edges coming out of each vertex. In Pseudo-Code 5.6 lines 4-5, the algorithm then initializes its solution with a set of *s* trips, one for each source, where each trip contains only that one source vertex. With these two elements, the G_{TA} algorithm enters into the while loop of Pseudo-Code 5.6 lines 6-11, that adds one unassigned target to the end of one of the trips in every complete iteration. This is done in a greedy fashion using the list of edges originating from the end vertex of each

trip, $Tr_i(v_{end}(E))$. The lowest weight outgoing edge, $e_{min,i} \in Tr_i(v_{end}(E))$, that connects to an unassigned target is compared for each trip, and the trip with the smallest $e_{min,i,}$ which will be referred to as e_{min}, is expanded to include that edge and the corresponding target. This target however is added provided that adding e_{min} and the target does not violate any of that source's trip constraints. If the constraints are violated, the next lowest weight outgoing edge connecting to an unassigned target for this trip is again compared with the other trip's lowest weight outgoing edges, a new e_{min} is determined and the process repeats.

The assignment "while loop" is continued until either 1) all of the targets are assigned to a trip or 2) all of the trips' outgoing edges, $Tr_i(v_{end}(E))$, are investigated and fail the constraint check process. In the first case, a complete solution is found and this solution along with its cost is returned. In the second case, no solution is found and a solution cost of infinity is returned. This second case is not an uncommon result in running greedy methods in general in heavily constrained problems as greedy methods have no way to recover from an early step that perhaps used considerable amounts of constrained resources for a small cost benefit. This highlights another advantage of this paper's new methods as since the new methods are based on the G^*_{TA} method, they are able to reconsider previous partial solutions that were not explored earlier for having slightly higher costs but may have resulted in less resource consuming solutions. The impact of constraints on G^*_{TA} methods has been discussed further in ref. [5.18], but in general is beyond the scope of this paper. This paper will only consider problems where the constraints do not influence the solution. This not only assures that the standard G_{TA} method will determine a solution, but helps to provide a conservative estimate of the H-G^*_{TA} performance. However, constraint

checks are still executed throughout all tests in order to report more accurate computation times for the general case.

The big "O" runtime of this algorithm is shown in equation (5.9) below. The O(|*S*+*T*||*T*|log|*T*|) term of equation (5.9) originates from the necessity to create a sorted list of edges to all targets, (|*T*|log|*T*|), coming out of each vertex, (|*S*|+|*T*|). The greedy assignment of targets to the sources is then O(|*T*||*S*||*T*|) where the first |*T*| term is for the iterations of Pseudo-Code 5.6 lines 6-11 where one target is assigned to a trip ever iteration. Also within every iteration, the best edge coming out of the end of each source's current trip to an unassigned target must be determined, which results in the |*S*| term. However an additional |*T*| term in the big "O" equation must be added because in determining that best edge to an unassigned target, it is possible that the entire list of edges coming out of the end trip vertex must be investigated. The overall big"O" equation of (5.9) is most accurately represented as the sum of the list formation and assignment big "O" terms as the dominant term can change depending upon the size of |*S*| and |*T*|.

$$O(|S+T||T|\log|T|)+O(|T||S||T|) \tag{5.9}$$

(*Incude*?: With the greedy standard established, which will be referred to as the Greedy benchmark, the goal set forth in performing the tests for this paper was to show that the H-G^*_{TA} new method could not only solve the problems in less computational time than the standard benchmark method but potentially scale better with respect to computational time while producing at least comparable if not better approximations to the MTSP problem.)

5.10 Implementation Test Setup

As this paper offers several variations to the H-G$^*_{TA}$ method, this section provides a description of a series of implementation tests developed with a specific focus for analyzing the runtime and relative percent error for each variation. The variations of the H-G$^*_{TA}$ method considered are outlined in Table 5.2 below. The Centroid H-G$^*_{TA}$ method is included to demonstrate the results of using a more traditional hierarchical approach; the Neighbor H-G$^*_{TA}$ method is included to show the benefits of using the new neighbor pair characterization introduced by this paper; and for completeness, the Mixed H-G$^*_{TA}$ method is included as a combination of the centroid based and neighbor based approaches. In Section 5.11, all three of these variations were run against the standard Greedy benchmark method, G_{TA}, discussed in Section 5.9.

Table 5.2: Algorithmic Components with Section Number References (horizontal axis) Incorporated into Variations of the H-G$^*_{TA}$ Method (vertical axis)

H-G$^*_{TA}$ method	Centroid Based Clustering (5.4, 5.5.1)	Neighbor Based Clustering (5,4, 5.5.3)	Centroid Based Sub-Problem TSPs (5.5.2)	Centroid Based Recursion (5.7.1)	Neighbor Based Recursion (5.7.2)	4-Opt Post Optimization (5.8)
Centroid	X		X	X		X
Neighbor		X	X		X	X
Mixed	X		X		X	X

The first H-G$^*_{TA}$ method variation, Centroid H-G$^*_{TA}$, follows a more traditional approach of using hierarchical cluster based methods. In all stages, this variation places a strong emphasis on the centroid as the representative characteristic for summarizing the properties of the original task allocation problem. Hence, the Centroid H-G$^*_{TA}$ variation follows the centroid based clustering of Section 5.4 and

5.5.1, the centroid sub-problem method of Section 5.5.2, the centroid recursion method of Section 5.7.1 as well as the Summary Problem method of Section 5.6 and the K-Opt post-optimization technique of Section 5.8.

The second H-G$^*_{TA}$ variation, Neighbor H-G$^*_{TA}$, takes a different approach by placing most of the algorithmic importance on the relationship between hierarchical clusters through the use of the cluster neighbor pairs. As shown in Table 5.2, this method uses the neighbor pair edge weights through neighbor based clustering, as described in Sections 5.5.3 and 5.5.4, and it uses the neighbor based recursion method of Section 5.7.2 to determine the final trips from the top level summary problem of Section 5.6. The only departure from using all neighbor based methods is in determining the cost and resource usage estimates from the cluster sub-problems. As Section 5.5.5 discussed, although the neighbor based sub-problem methods provided better cost and resource usage estimates, the neighbor based sub-problem methods take considerably more computation time. In addition, on average the cost and resource usage estimates produced by the neighbor based sub-problem solutions, as compared to those produced by the centroid sub-problem solutions, varied by less than 9%. Therefore, as only estimates are required at this point, the centroid based sub-problem TSPs proved to be a preferable alternative.

For completeness, the Mixed H-G$^*_{TA}$ variation is examined as well, which uses the traditional centroid based method for creating the clusters. Hence, the top level MTSP Summary problem also uses the centroid to centroid cluster costs of Section 5.5.1. However, the Mixed H-G$^*_{TA}$ variation also calculates the cluster neighbor pairs for use in obtaining the final trips from the Summary problem solution through neighbor based recursion.

In all three H-G^*_{TA} variations, however, the final trips from the recursion stage were run through 4-Opt post optimization, i.e. K-Opt where *kopt* is 4. The 4-Opt algorithm was chosen for its relative speed and its ability to improve cluster transitions, as discussed in Section 5.8.

The inputs to the tests of this study were setup to vary the number of sources, *s*, from {2..6}, as this range is of particular interest to several applications mentioned in Section 5.1, as noted in [5.18] and [5.30], and the number of targets from {50..250} at increments of 50. This range of targets is considered very large for task allocation target set studies; many studies, as described in Section 5.1, consider only a maximum target set size of no more than 50 targets. However, problems of this size can occur quite frequently in many of the applications cited in Section 5.1, such as those mentioned in [5.7] in particular.

As this study is largely concerned with computational runtime, it is assumed here that the required time threshold is 0.25 seconds, which is appropriate for a real-time application. This time threshold is also common to other studies, such as the work by John How's group in higher level task planning [5.10],[5.17],[5.32]-[5.40]. Therefore, in this study, any method that can solve these very large target set problems, while still providing solutions of quality that is comparable to those of the Greedy benchmark, will be considered a significant advancement. Due to this time threshold, both the maximum number of targets per cluster, $p_{c,max}$, and the maximum number of clusters allowed at the top level of the cluster hierarchy, $k_{top,max}$, inputs were set to 6 since the results presented in [5.18] demonstrated that problems of this size can be solved in a reasonable amount of time, in order to allow this study's largest

scale overall problems to be solved within the 0.25 second threshold time. This choice is validated further after the presentation of the results and the big "O" runtime analysis in Section 5.11.

All tests in this study were performed using a 2.0GHz processor PC with 2 GB RAM as this is a computer type was also used in past studies, [5.18],[5.30], which will allow for more direct comparison. The tests were run within the Cornell RoboFlag v2.1 testbed, which has been shown to be a high fidelity robotic simulation environment and has been used in numerous AFRL, Cornell, and Cal-Tech experiments over the past several years [5.32]-[5.40].

As discussed throughout this paper, the tests also use the intuitive 2D problem space, where the cost equation is simply the distance between vertices. This is expressed in equation (5.9) where $w_{i,j}$ is the weight of the edge between vertices i and j, as mentioned in Section 5.2, and $V_{i,x}$ and $V_{i,y}$ are the is the x-position and y-position of vertex V_i, respectively.

$$w_{i,j} = |(V_{i,x} - V_{j,x})| + |(V_{i,y} - V_{j,y})| \qquad \forall i \in V, j \in V \tag{5.9}$$

As part of utilizing this transition cost equation, the goal of determining solutions of comparable quality to those of the Greedy benchmark is considered to be achieved if the solutions from the H-G^*_{TA} variation are on average within 10% relative percent error from the Greedy benchmark. This is shown in equations (5.10) and (5.11) where J_{Greedy} is the cost of the final solution produced from the Greedy

benchmark, J_{HG^*TA} is the cost of the final solution produced by one of the H-G$^*_{TA}$ variations and $\%E_G$ is the relative percent error from the Greedy benchmark.

$$\%E_G = \left(\frac{J_{HG^*_{TA}} - J_{Greedy}}{J_{Greedy}} \right) \tag{5.10}$$

$$ave(\%E_G) \leq 10\% \tag{5.11}$$

As mentioned in Section 5.9, all tests also did not enforce any limiting constraints. This is because constrained tests can frequently lead to the Greedy benchmark failing to produce a feasible answer. Furthermore, to the authors knowledge a standard metric has not been established to measure how constrained a MTSP problem is. However as this study is concerned with runtimes, three constraints on individual source's fuel, target's time windows, and overall solution time were checked for all partial and final solutions within all H-G$^*_{TA}$ variations and all checks within all variations showed that the constraints were satisfied. Additional details on the influence of constraints on G$^*_{TA}$ based methods can be found in [5.30]. In short however, unlike in using MILP methods, constraints typically have the effect of shortening the runtime of G$^*_{TA}$ methods, as the constraints help to reduce the search space. Furthermore, in situations where a greedy based method may be forced to end without a feasible solution due to constraints, G$^*_{TA}$ methods is guaranteed to be able to "back-track" through exploring other parts of their search trees to determine a feasible solution if one exists.

The test problems are divided into two categories based on two approaches for target placement. The first approach used a uniform random distribution of both the

targets and the sources were within an 800 by 1200 area. Traditionally, A* based methods do not perform as well within these type of random environments while, as mentioned in Section 5.9, greedy based methods typically perform at their best in random environments. Hence, the use of random environments will also serves this to conservatively demonstrate the performance of the H-G*$_{TA}$ method.

The second approach of target and source placement used Gaussian distributions around a 6 randomly chosen locations within the problem space where target set was split evenly across these 6 locations. The standard deviations for the Gaussian distributions was constant for at all locations within the same test. The standard deviations were varied through from {25..100} at increments of 25 across each set of tests. However, neither the Greedy benchmark, nor the H-G*$_{TA}$ variations were given any information regarding these Gaussian standard deviations.

5.11 Implementation Test Results

5.11.1 Random Target Placement Test Results

The average results of running the Greedy benchmark and the H-G*$_{TA}$ variations on sets of 1500 randomly generated problems for every (*s*,*t*) pair are shown below in Table 5.3, and the semi-log graphs of Figures 5.13 and 5.14. Overall, it can be clearly seen from the results that all variants of the H-G*$_{TA}$ run on average significantly faster than the Greedy benchmark. The only case that the Greedy benchmark ran slightly faster was in the smallest source set, smallest target set case (*s*=2, *t*=50) where the base initialization time of the H-G*$_{TA}$ is significant enough to make the computational times comparable. However, in the larger scale problems, all H-G*$_{TA}$ variants are at least one order of magnitude faster than the Greedy benchmark; for the *t*=250 case the Centroid H-G*$_{TA}$ variant ran on average two orders of

magnitude faster. In all cases, the standard deviations of the average computation times of all methods were very close to their averages, which is a good result for many MTSP algorithms [3.18],[3.30]. These points also demonstrate that all variants of the H-G^*_{TA} method show improved scaling over the Greedy benchmark which is perhaps most clearly seen in Figures 5.13 and 5.14.

Table 5.3 : Implementation Test Results Comparing the Greedy Benchmark to Three Variations of the H-G*TA method with regards to Average Computation Time and Average Percent Error from the Greedy Benchmark Method under Random Target Placement Conditions

		Computation Time (milliseconds)				%Error Compared to Greedy, $\%E_G$		
s	t	Greedy	Centroid	Neighbor	Mixed	Centroid	Neighbor	Mixed
2	50	8.73	3.75	9.90	6.45	10.85	-1.50	-0.52
2	100	152.42	8.13	31.96	17.22	17.27	-1.52	0.38
2	150	566.20	13.82	73.48	30.52	20.34	-1.34	0.37
2	200	893.34	30.11	128.65	119.52	21.33	-1.70	-0.32
2	250	1665.89	41.68	148.64	160.09	21.82	-2.35	-0.95
4	50	28.35	15.01	18.00	14.04	15.49	5.74	4.92
4	100	426.36	27.19	56.74	28.08	20.29	3.77	4.25
4	150	732.73	28.25	80.00	57.78	22.91	3.51	3.94
4	200	1477.50	31.88	117.16	110.98	23.67	2.71	3.02
4	250	3163.78	41.61	191.44	255.46	23.97	1.27	2.22
6	50	108.58	52.45	32.85	27.19	18.87	10.52	8.98
6	100	679.29	53.68	76.84	48.43	22.08	7.92	7.05
6	150	1131.87	71.54	94.24	81.37	25.20	6.94	6.51
6	200	2268.18	78.47	143.40	159.17	25.54	5.92	5.65
6	250	4966.01	80.01	240.83	393.87	25.50	4.47	4.60

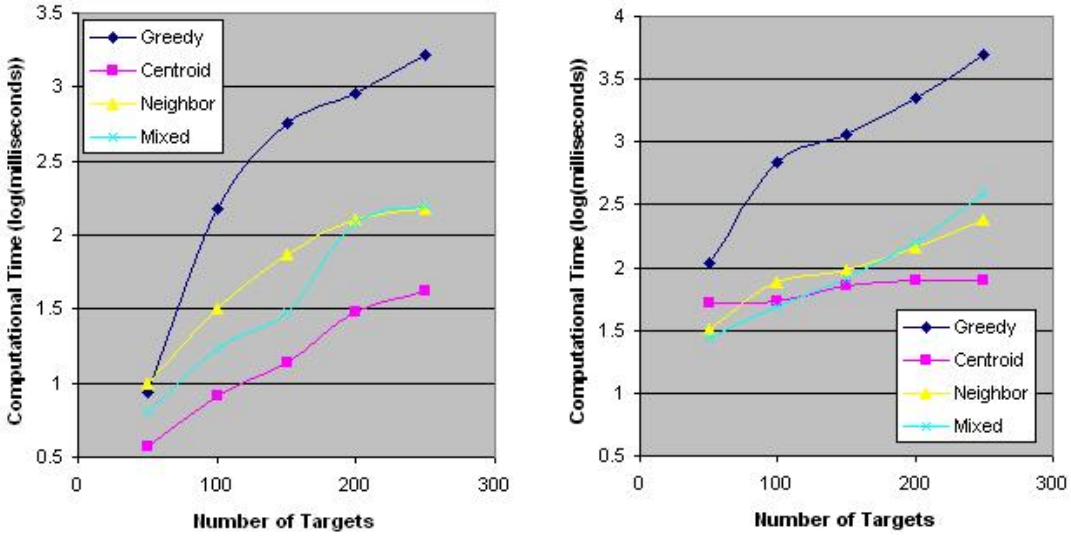

Figure 5.13 and 5.14: Average Computation Runtime of the Greedy Benchmark Method and the H-G$^*_{TA}$ Variations for Values of t = {50..250}, Uniform Randomly Distribution and left) s = 2, and right) s = 6

As greedy methods are traditionally known for their speed, these results may seem surprising. However a justification can be seen through the examination of the runtime of each of the components discussed in the previous sections. As explained in Section 5.9, the Greedy benchmark has a big "O" runtime of $O(|S+T||T|\log|T|) + O(|T||S||T|)$, where the first term is for determining sorted edge lists for all outgoing edges from all vertices and the second term is for the assignment of the targets to the source's trips. Both terms are at least polynomial with respect to $|T|$ which is the largest sized input set. Hence, for the large values of $|T|$ considered in this study, the computational runtimes can grow large fairly quickly.

Table 5.4 below provides a summary of the big "O" runtime analysis for the main components of the H-G$^*_{TA}$ method, which includes clustering, summarizing the clusters, solving the top level MTSP Summary problem, recursively solving the final

trip formation, and applying the K-Opt post-optimization. In order for the results presented in Table 5.3 to be obtained, it must follow that not only must each individual component be faster than the greedy standard but so must the sum of the times for these H-G$^*_{TA}$ components.

H-G$^*_{TA}$ Component	**Component Runtime Synopsis**
Clustering	$O(\|T_\alpha\|\|K_\alpha\|) + O(\|K_\alpha\|) \ \forall \alpha$
Neighbor Pair Creation	$O(\|K_\alpha\|^2(p_{c,max})^2)$
Sub-Problems	$O(\|K\| * (TSP(1,p_{c,max})))$
Summary Problem	$O(MTSP(\|S\|,k_{top,max}))$
Centroid Recursion	$O(\|K\| * p_{c,max})$
Neighbor Recursion	$O(\|K\| * (TSP(1,p_{c,max})))$
Post-Optimization	$O(_{kopt}P_{kopt} * \|T\|\text{-}kopt)$
where from ref. [5.18]	$TSP(1,p_{c,max})$ = 0.240ms, σ =0.0713ms $MTSP(\|S\|,k_{top,max})$ = 3.673ms, σ =5.2772ms

Table 5.4 Big "O" Runtime for Components of the H-G$^*_{TA}$ Method

The largest polynomial in the big "O" calculations of Table 5,4, comes from the $|T_\alpha||K_\alpha|$ term where $|T_\alpha|$ can be as large as $|T|$ in the lowest level of clustering but even at that level, K_α is up to $1/p_{c,max}$ smaller that $|T|$. Hence, the $|T_\alpha||K_\alpha|$ term is significantly smaller than the largest polynomials of the greedy standard. Furthermore, in the cluster iterations of higher hierarchical levels, both the $|T_\alpha|$ and $|K_\alpha|$ decrease by up to a factor of $(1 - {}^1/p_{c,max})$. Hence, higher level clustering occurs much faster. Additionally, previous studies on the cluster algorithm have shown that clustering sets of 50 points to 250 points, with $p_{c,max}$ = 6, $k_{top,max}$=6 as was done in this study, ranged in average times of 3 ms, σ=0.37ms to 72 ms, σ=4.1ms respectively [5.27].

Solving the cluster sub-problems at first may also appear to be computationally intensive as each sub-problem requires a TSP solution. However, using the G^*_{TA} method to solve TSPs at a maximum size of $p_{c,max}$ for this study resulted in an average solution time of 0.240ms with a standard deviation, σ, of 0.0713ms [5.18]. In the largest data sets, t=250, the average number of total clusters at all levels of the hierarchy was 73 with a standard deviation of 2.5. Hence, even at three standard deviations away in both the number of clusters and the TSP time, and assuming that all clusters contained the maximum number of members, the sub-problem time would be 36.5 ms.

These numbers also aid to verify the hypothesis that splitting a large NP-Hard problem into several smaller problems would result in faster computation times. The Summary problem also demonstrates the H-G^*_{TA} method's use of this hypothesis in that the Summary problem is a smaller MTSP with $s_{summary\ problem} = |S|$ but $t_{summary\ problem} = k_{top,max} << |T|$. Referring to the previous work of [5.18], MTSPs where, s=6, t=6, were solved on average in 3.673ms, σ =5.277ms. The standard deviation is somewhat large but even at three standard deviations away from the mean, the runtime for this step is just over 19.5ms.

The neighbor recursion component also requires that $|K|$ TSPs be solved, identical in scale to those of the sub-problems. Therefore, in the worst case of the assuming three standard deviations above the average mentioned in the sub-problem discussion, the maximum runtime would be just over 36.5 ms for the largest target case.

The post-optimization runtime is rather significant as well. Using 4-Opt, a single run of this method is O(16|*T*|). As 4-Opt is an iterative method that typically ran for 4-5 iterations in the centroid based recursion method case with t=250, this post-optimization runtime accounts for much of the remaining time in the overall averages reported in Table 5.3. However, as explained below, this additional time led to significant improvements in the solution accuracy of the Centroid H-G$^*_{TA}$ variation.

Additional time required, not explicitly detailed above, includes time required for creating neighbor pairs and edge cost calculation, as well as the setup of data at the beginning of and clearing data at the end of each component to conserve storage space. By themselves, only the neighbor pair creation is significant in terms of computation time, as was discussed previously in Sections 5.5.3 and 5.5.4. The combined runtime of the neighbor pair creation and the other algorithmic components are included of the average computational runtime summary of Table 5.3.

The difference in computation times between the Greedy benchmark and the H-G$^*_{TA}$ variants can also be explained by examining the algorithms from a higher level. Looking at the two parts of the big "O" runtime equations for the Greedy algorithm and Pseudo-Code 5.6 as presented in Section 5.9, the $|S+T||T|\log|T|$ component of the greedy method, for the s=6, t=250 tests problems, is equivalent to over a half of a million. Then for the next most computationally expensive part under the same test conditions, the $|T||S||T|$ component is equivalent to 375,000. When comparing this to less than 65,000 comparisons made for creating the neighbor pairs, which is the largest value obtainable from the big "O" equations of Table 5.4, it begins to become understandable why the H-G$^*_{TA}$ method runs that much faster. Following the same logic, although solving even a small MTSP may intuitively seem too time

consuming, solving the {s=6, t=6} MTSP of the Summary problem requires only 500 node expansions on average [5.18]. Granted, a single run of the node-expansion process is more time consuming than a single run of any sub-component of the Greedy benchmark. However, as the Table 5.3 data verifies, the significant difference in the number of times the components of the Greedy benchmark must run as compared to the number of times that the H-G^*_{TA} components must run more than makes up for the difference in time of running any single component of the two methods.

Comparing each of the H-G^*_{TA} variations using Table 5.3 and Figures 5.13 and 5.14, it is clear that the Centroid H-G^*_{TA} method runs the fastest. This is not surprising as it does not require the extra time to determine the neighbor pairs, and more importantly, does not need to solve additional TSP problems in its final recursion. These lower computation times however come at the cost of a significant increase in the percent errors relative to the solutions of the Greedy benchmark. The Neighbor H-G^*_{TA}, in contrast, is able to produce comparable solutions to the Greedy benchmark while still running considerably faster. In some of the larger test cases the Neighbor H-G^*_{TA} variations runs up to an order of magnitude faster than the Greedy benchmark, even in these random tests where the H-G^*_{TA} method should perform at its worst. The Mixed H-G^*_{TA}, although in most cases would run slightly faster than the Neighbor H-G^*_{TA}, produced solutions of a quality that were only slightly worse than the Neighbor H-G^*_{TA} but still significantly better than the Centroid H-G^*_{TA}. This shows that, since the neighbor based recursion is the only component that is different between the Mixed and Centroid H-G^*_{TA} methods, the neighbor based recursion improves the overall solution quality on average by ~10% to 20%.

Neighbor H-G*$_{TA}$ produces better solutions than Mixed H-G*$_{TA}$. Since the neighbor pairs must be determined in order to run the neighbor based recursion component, using the neighbor pair information in the clustering as well is of little consequence to the computational speed in return for improved solution quality. In addition, s shown in Table 5.3, Mixed H-G*$_{TA}$ does not scale quite as well as Neighbor H-G*$_{TA}$ with respect to computation runtimes. The reason for is attributed to the fact that the final solutions resulting from the required recursion in the Mixed H-G*$_{TA}$ variation took significantly more time in the post-optimization than the Neighbor H-G*$_{TA}$ variation.

In general, the relative solution cost percent errors are larger for the H-G*$_{TA}$ variations when the number of sources increases. This can be expected given the discussion in Section 5.9 on how the greedy method solutions improve as the number of sources increases. One of the more interesting trends, however, is that the relative percent errors of the Neighbor and Mixed H-G*$_{TA}$ variations tend to decrease with an increase in the number of targets in these random tests. This trend is seen in part from the ratio of targets to sources being larger in these cases. Increases in this ratio, in general, have an adverse effect on the Greedy benchmark, while the hierarchical nature of H-G*$_{TA}$, on the other hand, makes it less susceptible to changes in the targets to sources ratio .

The number of sources also has an important effect on the scaling of the H-G*$_{TA}$ variations computational runtime. The most obvious effect is seen in the upward shift of the graph lines shown from Figure 5.13 to Figure 5.14. More interestingly, for larger values of $|S|$, both the Neighbor and the Centroid H-G*$_{TA}$ variations show improved scaling with respect to increasing $|T|$, as can be seen by the generally

"flatter" lines. Although the number of sources does not influence the clustering algorithmic components, this improved scaling effect can be attributed to the fact that with more sources, the average length of the any source's trip decreases. This in turn can reduce both the recursion runtimes and more importantly, especially in the Centroid variation, the post-optimization runtimes. The Mixed H-G^*_{TA} does not exhibit this trend, however, because the centroid based clustering does not create clusters that coincide well with the neighbor based recursion. Hence, the improvement seen in the other variations in the recursion and post-optimization is not evident in the Mixed variation.

Considering the goal of a 0.25 second overall computational runtime bounds and the comparable solution quality goal of being within 10% of the Greedy benchmark, the Neighbor H-G^*_{TA} variation shows the best combination of improved computation speed while still showing highly respectable solution quality compared to the Greedy Benchmark. Hence, when applying the Neighbor variation, the H-G^*_{TA} method has been shown to be a viable method for solving very large task allocation problems for real-time applications.

5.11.2 Gaussian Clusters Results

The Gaussian cluster tests results for running the Greedy benchmark against the H-G^*_{TA} variations are shown below for four different cases. In these four cases, 6 Gaussian clusters were formed from a total of *t* targets, where the standard deviation for each case was varied from $\sigma = \{25...100\}$ at increments of 25.

Table 5.5 : Implementation Test Results Comparing the Greedy Benchmark to Three Variations of the H-G*TA method with regards to Average Computation Time and Average Percent Error from the Greedy Benchmark Method under Gaussian Target Placement Conditions

σ 25		Time (milliseconds)				%Error Compared to Greedy		
s	*t*	Greedy	Centroid	Neighbor	Mixed	Centroid	Neighbor	Mixed
2	50	16.00	4.06	9.65	9.55	7.41	-3.71	-2.24
2	150	560.75	16.84	62.81	53.47	9.95	-5.30	-3.59
2	250	2159.58	36.13	203.91	162.95	17.79	-0.69	1.43
4	50	64.55	7.34	14.63	16.34	9.74	-1.90	-0.25
4	150	1178.26	24.49	80.60	72.03	10.71	-4.13	-2.65
4	250	4464.10	45.98	251.71	228.59	18.89	0.96	2.88
6	50	129.42	16.92	18.78	27.71	10.86	-0.81	0.98
6	150	1801.67	45.58	93.92	106.04	12.15	-2.93	-1.30
6	250	6670.40	63.09	294.41	292.17	19.94	1.93	3.89
σ 50		Time (milliseconds)				%Error Compared to Greedy		
s	*t*	Greedy	Centroid	Neighbor	Mixed	Centroid	Neighbor	Mixed
2	50	13.63	4.16	9.76	6.68	5.86	-5.18	-3.77
2	150	497.23	16.62	56.11	32.62	11.69	-4.95	-1.98
2	250	2327.01	35.51	175.55	88.77	17.47	-1.51	2.97
4	50	64.41	7.48	13.63	9.91	7.95	-2.28	-1.18
4	150	1032.51	24.17	65.87	40.65	13.05	-2.66	0.14
4	250	4851.02	45.94	208.58	107.98	19.04	0.28	4.71
6	50	125.61	18.13	17.11	13.80	9.74	-0.21	1.11
6	150	1520.61	38.99	74.06	49.44	14.38	-0.82	2.03
6	250	7521.31	62.38	239.04	129.73	20.37	2.24	7.00
σ 75		Time (milliseconds)				%Error Compared to Greedy		
s	*t*	Greedy	Centroid	Neighbor	Mixed	Centroid	Neighbor	Mixed
2	50	6.03	4.03	10.55	9.18	5.52	-5.17	-3.89
2	150	497.52	15.21	59.58	49.12	12.98	-4.38	-2.90
2	250	2586.68	32.34	195.30	163.36	18.24	-2.41	-0.87
4	50	55.39	7.61	13.67	17.07	8.74	-1.16	-0.33
4	150	1057.94	22.81	72.99	79.47	15.57	-0.90	-0.02
4	250	5215.87	42.30	235.45	267.08	20.32	0.36	1.56
6	50	128.07	21.05	17.19	32.75	10.92	2.01	2.29
6	150	1565.82	45.54	85.80	117.97	16.84	1.52	1.89
6	250	6523.51	52.44	263.52	331.23	21.50	2.39	3.04
σ 100		Time (milliseconds)				%Error Compared to Greedy		
s	*t*	Greedy	Centroid	Neighbor	Mixed	Centroid	Neighbor	Mixed
2	50	5.74	3.95	10.29	9.34	6.83	-3.99	-2.45
2	150	545.82	14.97	64.77	53.57	15.53	-2.73	-1.30
2	250	2353.57	31.73	170.92	160.20	19.76	-1.75	-0.46
4	50	52.18	8.73	13.78	19.29	10.88	1.12	1.77
4	150	1230.04	25.59	82.44	96.73	17.96	1.11	1.64
4	250	4650.39	42.41	225.49	270.86	22.15	1.53	2.34
6	50	131.50	29.40	19.41	41.32	13.16	4.79	4.69
6	150	1863.65	59.45	98.16	150.10	19.84	3.80	3.92
6	250	6901.99	71.42	271.61	370.93	23.27	3.64	4.15

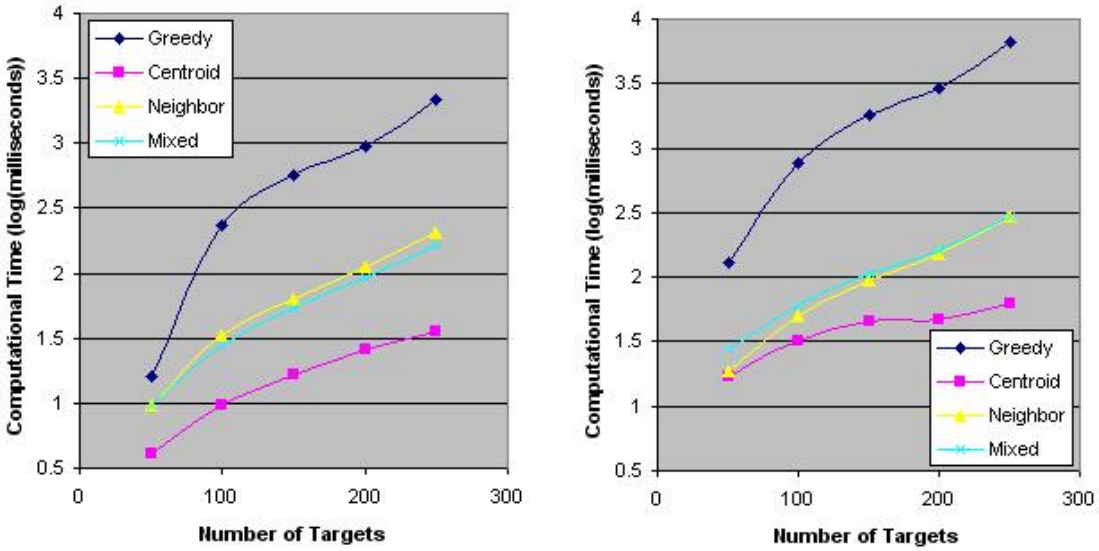

Figure 5.15 and 5.16: Average Computation Runtime of the Greedy Benchmark Method and the H-G$^*_{TA}$ Variations for Values of t = {50..250}, Gaussian Distribution σ=25, and left) s = 2, and right) s = 6

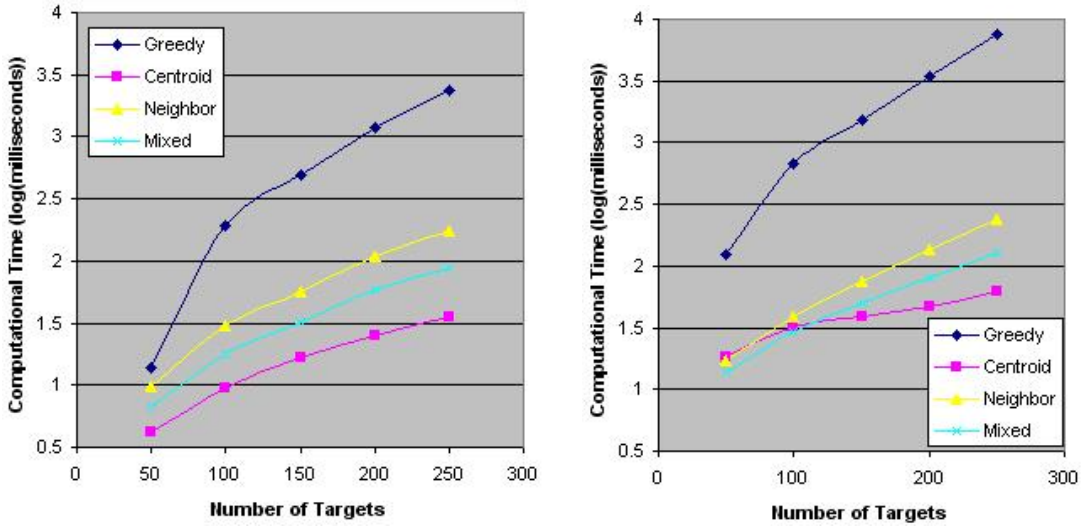

Figure 5.17 and 5.18: Average Computation Runtime of the Greedy Benchmark Method and the H-G$^*_{TA}$ Variations for Values of t = {50..250}, Gaussian Distribution σ=50, and left) s = 2, and right) s = 6

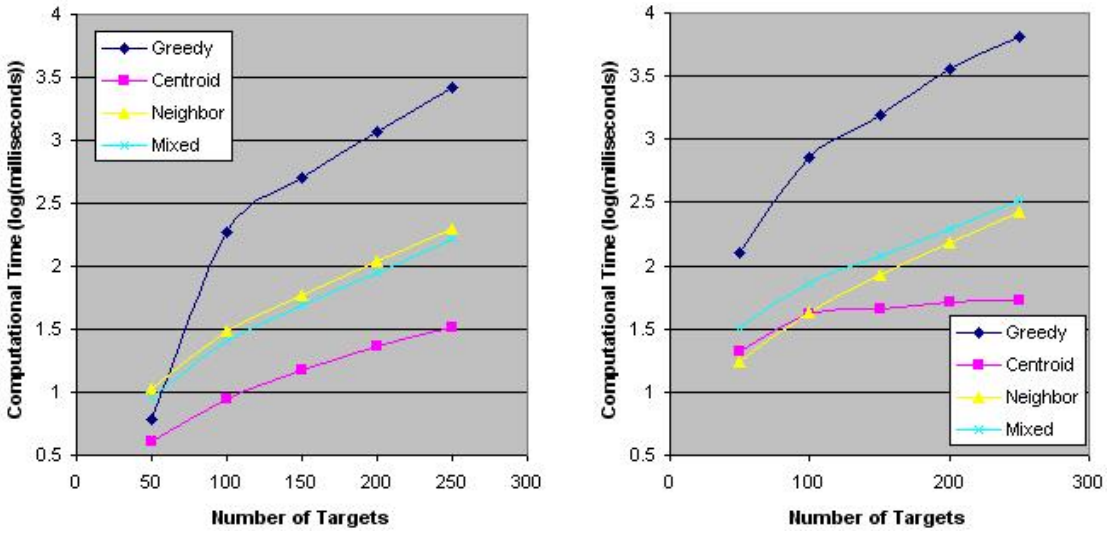

Figure 5.19 and 5.20: Average Computation Runtime of the Greedy Benchmark Method and the H-G*$_{TA}$ Variations for Values of t = {50..250}, Gaussian Distribution σ=75, and left) s = 2, and right) s = 6

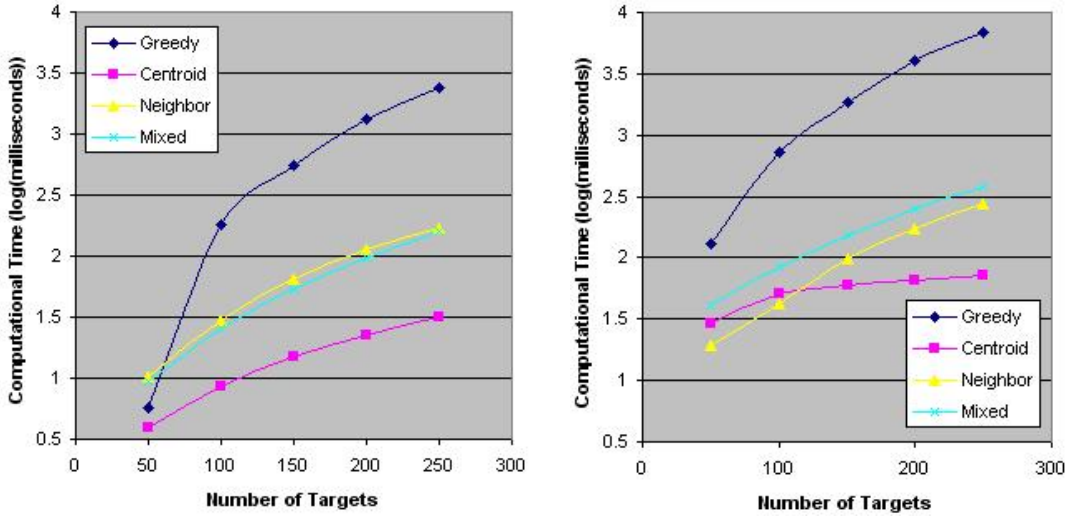

Figure 5.21 and 5.22: Average Computation Runtime of the Greedy Benchmark Method and the H-G*$_{TA}$ Variations for Values of t = {50..250}, Gaussian Distribution σ=100, and left) s = 2, and right) s = 6

Overall, the results of these tests showed that most of the same trends found in the random target tests were found again, particularly across sets of tests with the same Gaussian cluster standard deviation. The most immediate difference, however, is that in all these tests, all H-G^*_{TA} variations had improved average relative percent errors over the random tests. This was expected, however, as applying a clustering method to a problem that naturally has a clustered form will intuitively result in the formation of sub-problems that better represent the original problem.

Also with respect to tests with the same Gaussian cluster standard deviation, the percent errors are best at the smallest problem sizes of t=50. This is because, as the Gaussian distribution problems become larger, the cluster hierarchies become taller, and the more these problems begin to lose their clustered form which results in these Gaussian distribution problems beginning to resemble random problems.

As the standard deviation of the Gaussian distributions increases, in general however, the problems more closely resemble the random case, resulting in errors of a similar size, The computation times also tend to increase as the Gaussian clusters' standard deviation increases since, for those problems that initially have a tightly clustered form to begin with, the H-G^*_{TA} clustering component can converge in fewer iterations.

One departure from these trends can be seen in the case when the Gaussian clusters' standard deviation is 25. In this case, the Gaussian distributions are relatively tight compared to their relative center locations, and the smallest costs between Gaussian distributions can be significantly larger than the largest costs between any two members within a Gaussian distribution. In this case, when the Greedy benchmark

is applied, once a source "travels" to a Gaussian distribution the Greedy benchmark often assigns all of that Gaussian distribution's targets to that source. This causes the overall problem to appear to the greedy method as a much smaller scale problem where each Gaussian distribution target is simply a "super target". Outliers within each Gaussian distribution prevent this from occurring exactly, but overall both the Greedy benchmark and H-G^*_{TA} method are able to benefit from the problems' structure.

In order to see better results from the H-G^*_{TA} method for task allocation problems with an inherent tight clustering of targets, such as the Gaussian distribution σ=25 case, the maximum distortion of the clustering algorithm, D_{max}, must be made smaller to adjust for the tightness of the inherent clusters. In the tests presented in this section, however, neither D_{max}, nor any of the clustering input parameters were adjusted, so that a more direct comparison could be made between the different Gaussian distribution standard deviation cases.

This last case exemplifies the important point that having even a small amount of additional information about the structure of a problem can lead to significant improvements in the performance of H-G^*_{TA}. Even in this simple case, where the rather small additional structure provided by the Gaussian distributions was added, and the input parameters were not tuned to exploit the structure, there still was an observable improvement over the unstructured random case in nearly all tests, with up to an average $\%E_G$ improvement of nearly 11% in the s=6, t=50, Gaussian σ=50 versus the s=6, t=50 random distribution case.

5.12 Conclusions

The H-G^*_{TA} method has been established as a fast and viable approximation method for large scale real-time task allocation problems, capable of producing comparable quality solutions in computation times that are up to an order of magnitude faster than a standard greedy approximation method while using the Neighbor variation of H-G^*_{TA}; computation time of up to two orders of magnitude faster while using the Centroid variation of H-G^*_{TA}. The incorporation of the neighbor pair cluster characterization in both the hierarchical clustering, and in the recursive final solution creation, over using a traditional centroid based approach, was shown however to significantly improve the solution quality for reasonable increases in computation time. Finally, the performance of the new H-G^*_{TA} method was verified through a series of implementation tests using both random and Gaussian target distributions.

REFERENCES

[1] Campbell, M., "Planning Algorithm for Multiple Satellite Clusters," *Journal of Guidance, Control and Dynamics*, Sept-Oct 2003.

[2] G. Thomas, A. M. Howard, A. B. Williams, and A. Moore-Alston, "Multi-robot task allocation in lunar mission construction scenarios," in *IEEE International Conference on Systems,* Oct 2005

[3] Chandler, P., Pachter, M., *et al.* "Distributed Control for Multiple UAVs with Strongly Coupled Tasks," AIAA *Guidance, Navigation, and Control Conference*, August 2003

[4] Bellingham, J., Tillerson, T., Richards, A., How, J., "Multi-Task Allocation and Path Planning For Cooperating UAVs" *Conference on Coordination, Control and Optimization*, Nov. 2001

[5] Purwin O., D'Andrea R.: "Cornell Big Red 2003", in: Polani D., Bonarini A., Browning B., Yoshida K. (Eds), *Robocup 2003: Robot Soccer World CupVII, Lecture Notes in Artificial Intelligence,* Springer, Berlin, 2003

[6] Gerkey, B., Mataric, M., "A formal analysis and taxonomy of task allocation in multi-robot systems" *International. Journal of Robotics Research* 23(9):939-954, September 2004

[7] D. Schneider, A. Hoffman, C. Edmunds, B. Medina, J. Hosler "Adaptive Sensor Fleet Development of Inexpensive Multi-Agent Robotic Testbeds Using the NASA Multi-Purpose Exoterration for Robotic Studies" *National Aeronautics and Space Administration Internal Code 588*

[8] D. Dyke Weatherington, "DoD UAV Roadmap", *U.S. Department of Defense*, 2003.

[9] Richards A, Bellingham J, Tillerson M, and How J, "Coordination and Control of Multiple UAVs", *Guidance Navigation and Control Conference*, Aug. 2002.

[10] Y. Kuwata, A. Richards, T. Schouwenaars, and J.How, "Distributed Robust Receding Horizon Control for Multi-vehicle Guidance" *IEEE Transactions on Control Systems Technology Journal*, 2007

[11] C. Schumacher, P. Chandler, M. Pachter, and L. Pachter, "UAV Task Assignment with Timing Constraints via Mixed-Integer Linear Programming", *AIAA 3rd "Unmanned Unlimited" Technical Conference, Workshop and Exhibit*, September 2004

[12] M. A. Darrah, W. Niland, and B.M.Stolarik, “Multiple UAV Dynamic Task Allocation Using Mixed Integer Linear Programming in a Sead Mission,” in *Infotech@Aerospace*, Arlington, Virginia, September pp.26-29, 2005.

[13] N. Atay, and B.Bayazit “Mixed-Integer Linear Programming Solution to Multi-Robot Task Allocation Problem” *IEEE International Conference on Robotics and Automation*, 2007

[14] A. Bender. MILP based task mapping for heterogeneous multiprocessor ystem” *In Proceedings of EURO-DAC*, September 1996.

[15] A.Davare, J.Chong, Q. Zhu, D. Densmore, A. Sangiovanni-Vincentelli, “Classification, Customization, and Characterization:Using MILP for Task Allocation and Scheduling” University of California Berkley, Technical Report No. UCB/EECS-2006-166, December 2006.

[16] M. G. Earl and R. D'Andrea, “Iterative MILP Methods for Vehicle Control Problems,” *IEEE Conference on Decision and Control*, Atlantis, Paradise Island, Bahamas, Dec. 2004

[17] Richards, A., Kuwata, Y., How, J., “Experimental Demonstrations of Real Time MILP Control” *AIAA Guidance Navigation and Control Conference*, Aug. 2003.

[18] D. Schneider and M.Campbell “Improved Optimistic Predictive Cost Method For Faster G*TA Real-Time Task Allocation” *Paper submitted to IEEE Transactions on Robotics*

[19] P. B. Sujit, A. Sinha, and D. Ghose, “Multi-uav task allocation using team theory,” in *IEEE International Conference on Decision and Control, and the European Control Conference*, Seville, Spain, December 12-15 2005, pp. 1497–1502.

[20] R. Zlot and A. Stentz, “Complex task allocation for multiple robots,” in *International Conference on Robotics and Automation*, Barcelona, Spain, April 2005, pp. 1515–1522.

[21] B. P. Gerkey and M. J. Matari´c, “Sold!: Auction methods for multirobot coordination,” *IEEE Transactions on Robotics and Automation*, vol. 18, no. 5, pp. 758–786, October 2002.

[22] R. Zlot, A. Stentz, M.B. Dias, and S. Thayer, "Multi-robot Exploration Controlled by a Market Economy" *IEEE International Conference on Robotics and Automation*, 2002

[23] T. Lemaire, R. Alami, and S. Lacroix, "A distributed tasks allocation scheme in multi-uav context," *IEEE International Conference on Robotics and Automation*, New Orleans, LA, April 2004, pp. 3822–3827.

[24] M.B. Dias, R.M. Zlot, N. Kalra, and A. Stenz, "Market-Based Multirobot Coordination: A Survey and Analysis," *Proceedings of the IEEE,* Vol. 94, No. 7, July 2006

[25] Ousingsawat, J., and Campbell, M., "Multiple Vehicle Team Tasking for Cooperative Estimation", *American Control Conference*, 2004

[26] S. Rathinam, and R. Sengupta, "Lower and Upper Bounds for a Multiple Depot UAV Routing Problem", *IEEE Conference on Decision and Control*, December 2006

[27] D. Schneider and M.Campbell "Constrained Size Adaptive Hierarchical K-Means Clustering for Sub-Problem Division of NP Hard Problems" *Paper submitted to IEEE Transactions on Robotics*

[28] Parker, L. E, "ALLIANCE: An architecture for fault tolerant multi-robot cooperation", *IEEE Transactions on Robotics and Automation* 14(2), 220–240. 1998

[29] Korte, B. & Vygen, J., *Combinatorial Optimization: Theory and Algorithms*, Springer-Verlag, Berlin 2000

[30] D. Schneider, M. Campbell, "Real Time Optimal Task Allocation in Highly Dynamic Environments" *International Mechanical Engineering Congress and Exposition*, Nov., 2005

[31] Christian Nilsson, "Heuristics for the Traveling Salesman Problem", *Technical Paper of Linkoping University*, 2003

[32] Chaudhry, R. D'Andrea, and M. Campbell, "RoboFlag - A framework for exploring control, planning, and human interface issues related to coordinating multiple robots in a realtime dynamic environment", *Intl. Conf. on Robotics and Automation*, 2003

[33] J. Sullivan, S. Waydo and M. Campbell, "Using Stream Functions to Generate Complex Behavior" 2003 *Guidance, Navigation and Control Conference*.

[34] M. G. Earl and R. D'Andrea, "A study in cooperative control: The RoboFlag Drill,'" *Proceedings of the American Control Conference*, Anchorage, Alaska 2002

[35] Squire P.N., Galster, S.M., & Parasuraman, R. "The effects of levels of automation in the human control of multiple robots in the RoboFlag simulation environment." *Proceedings of the Second Human Performance, Situation Awareness, and Automation Conference*, 2004

[36] Veverka, J. and Campbell, M., "Experimental Study of Information Load on Operators in Semi-Autonomous Systems," *AIAA Guidance, Navigation and Control Conference*, Austin TX, Aug. 2003.

[37] M. Campbell, F. Bourgault, S. Galster, D. Schneider "Probabilistic Operator-Multiple Robot Modeling Using Bayesian Network Representation" *IEEE International Conference on Robotics and Automation,* April 2007

[38] R. D'Andrea and M. Babish, "The RoboFlag Testbed", *Proc. of the American Controls Conference*, June, 2003,

[39] D. Schneider, "The RoboFlag Website," *Cornell University*, October 2003, http://roboflag.mae.cornell.edu/

[40] Campbell, M., D'Andrea, R., Schneider, D., Chaudhry, A., Waydo, S., Sullivan, J., Veverka, J., Klochko, A., "RoboFlag Games using Systems Based, Hierarchical Control," *American Control Conference*, June 2003.

CHAPTER 6

ACTIVE LEARNING AND ASSESSMENT WITHIN THE NASA ROBOTICS ALLIANCE CADETS PROGRAM

In response to the 2006 *National Defense Education and Innovation Initiative*, NASA and DAVANNE LLC have collaborated to create the NASA Robotics Alliance Cadets Program to develop highly integrated and interactive STEM undergraduate curriculum. This paper investigates the NASA Cadet's use of Active Learning to not only meet the nationally recognized need for a formal assessment standard, but also ensure the sustainability of the program. To demonstrate the program's Active Learning tools' wide accessibility and their integration with the program's methodologies, this paper examines the NASA Cadets' robotics platform and its use within an educational experiment co-developed by Cornell University.

6.1 Motivation

Active Learning has been well established as an excellent method for increasing academic achievement, promoting the higher levels of Bloom's Taxonomy [6.1], developing supportive relationships among students and between students and teachers, and even improving students' attitudes towards STEM (Science, Technology, Engineering, and Mathematics) fields [6.2],[6.3]. These benefits, combined with the motivation provided by ABET's Engineering Criteria 2000, have inspired the development of numerous specialized programs that incorporate Active Learning at several of our nation's top colleges [6.4]-[6.8]. In fact, several top research groups, such as Dr. Felder's team at North Carolina State, have gone as far as to

provide the research equivalent of "how to" guides for incorporating Active Learning [6.8]. Despite these efforts, however, the need to extend these programs to more curriculums and more colleges continues to be voiced at a national level in such high profile documents as the 2003 *NASA Education Enterprise Strategy* and, more recently and quite strongly, in the 2006 *National Defense Education and Innovation Initiative* report [9],[10].

A significant reason that this continues to be a key issue is that although the energy and talent dedicated to creating existing programs has been in some cases remarkable, propagation of these developed programs is severely hindered by four main factors. First, many programs are too specific to a particular school's resources to be transferable. Second, the developed Active Learning tools often lack the flexibility to be utilized in other similar educational situations. Third, these programs do not include adequate partnered assessment strategies to ensure that the intended goals are being met, and, fourth, these programs commonly do not include plans for sustaining the program or adapting it to future student and teacher needs. Both the *NASA Education Enterprise Strategy* and the *National Defense Education and Innovation Initiative* offer evidence of these factors and emphasize in particular the need to do more to aid the frequently untargeted "Underrepresented and Underserved" student populations [6.9],[6.10].

As a result, a significant duplication of efforts has occurred across many institutions, with each developing its own innovative programs, Active Learning exercises, and even entire methodologies to teach the same material, but without having any method of objectively comparing their effectiveness. Although the current efforts should continue, they would be far more productive, effective and able to

advance the overall community's program quality if a standard for assessing the learning outcomes was established to highlight the strengths of each program This concept is echoed by several bodies of research that state that if methodologies are to be created by any organization, there must be assessment methods in place to determine the methodologies' effectiveness [6.10]-[6.14].

In response to this need, the NASA Robotics Alliance Cadets Program was designed around the concept of developing and incorporating assessment techniques that could be easily used to not only assess its own program effectiveness but could be easily incorporated into outside agencies' programs. This in turn can then provide a fair and balanced measure of assessing any program's ability to meet a similar set of learning objectives [6.15]. The development of the NASA Cadets Program's assessment suite also has the support of ASEE's Educational Research and Methods (ERM) Division; ASEE Executive Director Frank Huband has expressed his support of this program in "enhancing the effectiveness of other programs." [6.16]. This collection of assessment tools also fulfills the ABET recognized need to enable less experienced instructors to perform accurate measures of the quality of their own programs and Active Learning exercises. As ABET acknowledged in a similar discussion in 2006 regarding the use of outcomes-based methodologies, "It is apparent that while the new, outcomes-based criteria finally provide the opportunity for innovation and program individuality, they also appear to leave much interpretation open to program evaluators and faculty, many of whom, the constituents believe, have varying levels of sophistication and training in outcomes assessment." [6.17]

Once a program's strengths are identified, the NASA Cadets Program is dedicated to providing fellow colleagues and developers detailed implementation

procedures that can be used to ensure that the results can be reproduced across various intuitions. Moreover, NASA Cadets Program asserts that this is a necessary step that all leading educational facilities should follow in order to allow other institutions to quickly take advantage of these efforts and rapidly improve upon the educational quality of their own programs. Given the benefits of Active Learning stated earlier, the NASA Cadets Program holds this as a key practice in Active Learning and overall program development that must be adopted on a larger scale to meet the *National Defense Education and Innovation Initiative's* core goal to: "identify and promote best practices and programs in undergraduate STEM education, especially those that address college freshman attrition and under-representation of minorities and women in STEM fields." [6.9],[6.10],[6.15]

This paper offers an introduction to the new assessment standardization work being conducted by the NASA Cadets Program as part of a formal educational report currently being developed. This paper also examines the significant role that Active Learning plays in providing both in class assessment to instructors and students as well as in aiding instructors in conducting post-session assessment within this new standard. Finally, this paper discusses the educational capability of Active Learning tools within the program to demonstrate the utilization and wide applicability of this new assessment standard.

In order to provide the reader with background on the NASA Cadets Program, the paper begins with a condensed description of the program's methodologies for reaching its goals with particular attention given to provide an overview of the new assessment suite. This then motivates Section 6.3 to discuss the role of Active Learning as an assessment tool as well as some of the additional educational benefits

that can be achieved simultaneously. For completeness, Section 6.3.1 then provides a further description of the assessment suite with regards to methods by which the Learning Objectives of the Active Learning tools are themselves assessed. This section and the next section also describe the attention given to the targeted evaluation areas of critical thinking, innovation, troubleshooting and community. These are areas that extend beyond the traditional ABET focus of breadth, depth, and professionalism but have been identified as highly important if not crucial areas by the educational research community [6.18]-[6.20].

After establishing the assessment suite concepts in Section 6.3, Section 6.4 addresses accessibility and cross-institution applicability with regards to the incorporation of Active Learning into more equipment intensive settings such as labs and design projects. Section 6.4 also focuses on the NASA Cadets' robotic platform as a key element to achieving this goal and the platform's role in aiding in the new assessment standard. Then finally, Section 6.5 discusses a module developed for Cornell University in order to demonstrate the integration of many of the program aspects that have been emphasized throughout the paper.

6.2. The NASA Robotics Alliance Cadets Program

The NASA Robotics Alliance Cadets Program was created in September 2005 to develop a nationwide initiative to re-design the first two years of Mechanical Engineering, Electrical Engineering, and Computer Science as highly interactive and integrated curriculum. Furthermore, through these curriculum NASA would not only combat STEM attrition trends and diversity issues but ultimately inspire more students to pursue STEM careers while guaranteeing improved academic performance and knowledge retention [6.9],[6.10],[6.15].

At the heart of the NASA Cadets Program's core deliverables in realizing this goal is the NASA Cadets Instructor's Manual. The Instructor's Manual is a collection of detailed lesson plans that, in addition to outlining the core concepts and equations that are traditionally taught, it includes detailed implementation procedures for Active and Cooperative Learning techniques, planned discussions on evaluation methodologies and applications, and presenting real world motivations. Combined with carefully constructed homeworks and labs, together these lesson plans ultimately move engineering education beyond merely the Knowledge and Comprehension levels of Bloom's Learning Taxonomy that most current first and second year courses are limited to, into the higher levels of Analysis, Synthesis and Evaluation [6.1].

In order to make this leap possible, coupled with the NASA Cadets Instructor's Manual is a newly designed robotic platform. This platform was specifically created to allow a variety of Active Learning and other educational activities to be easily realizable across numerous institutions of varied resources. In fact, this platform is designed to be a highly robust yet modifiable testbed that is of low enough cost to allow every student to own their own robot. Given the robot's modular nature, students are then able to employ their courses' material in a very hands-on, results oriented setting and they are even encouraged to devise their own experiments to answer design problems. As the field of robotics requires expertise in all three target fields (Mechanical Engineering, Electrical Engineering, and Computer Science), required weekly interaction with the robotic platform will re-enforce the cross-course connections and will continually review older concepts while relating them to new material. A summary on the details of the robotic platform as and its use as an Active Learning and overall educational tool is provided in Section 6.4.

The design of creating the entire program to be as inexpensive as possible is actually crucial for the program to obtain its higher goals. Although it is certainly a requirement that the educational components developed be at, if not above, the standards of the country's highest regarded institutions, it is equally important that the program and the implementation procedures be as accessible and realizable as possible to even junior colleges nationwide. This objective relates back to the NASA Education Enterprise Strategy identification of the commonly untargeted Underrepresented and Underserved student populations within STEM fields [6.9]. Since the NASA Cadets's Program is centered on the first two years, it also offers the opportunity to develop student transfer programs from 2 year to 4 year schools that would have a better chance of reaching these populations. However, in order for these programs to be successful, the 2 year schools must first be able to afford to incorporate the NASA Cadets Program into their programs. Steps have already been taken to ensure that the NASA Cadet's Instructors Manual can be easily obtainable through the NASA Robotics Curriculum Clearinghouse (RCC) a currently well established, NASA administered on-line service that provides robotics related curriculum materials to educator members at low or no cost. Furthermore, the DAVANNE LLC, is dedicated to providing the program with a fully autonomous base robot at a cost of approximately $450, a price which equates to less than a textbook per course in a projected base 6-course program.

As part of integrating the Active Learning and robotics platform components into the lesson plan curriculum, the program is also designed around the need to incorporate effective assessment strategies from the beginning. The assessment methodology is detailed further in Section 6.3 but is overviewed briefly here. In

addition to following the accreditation rules and guidelines set forth by ABET, the educational model of Learning Objectives was chosen to aid in both the efficient design of NASA Cadet's Program courses as well as their assessment and comparison with current undergraduate courses. In short, the Learning Objective model states that all instructional goals will be phrased in the form "Given X, students will be able to perform Y, whose quality will be determined based on rubric Z". By providing both students involved with the NASA Robotics Alliance courses and those students who are instructed via more common methods with the same problems and information, i.e. "X", the students can then be asked to perform "Y" and can be measured and compared by the same standard "Z".

This in effect builds into the system a direct measure of student performance and can be easily incorporated into knowledge gain tests. Indirect measures such as student/faculty surveys and feedback interviews as well as student employment/further education trends will also be used to judge the quality of the program. Just as importantly, the program will also include newly developed tools for 'intangible' student assessment in vital engineering skill areas such as troubleshooting, innovation, design, community, and project management which have been traditionally overlooked.

As it is unrealistic to assume that the entire program would be instantly welcomed and adopted by every institution, the lesson plans developed by the NASA Cadets Program are developed to be highly modular in nature. This allows instructors the flexibility to integrate elements at a pace they deem reasonable. Furthermore, the NASA Cadets Program is designed to allow participating instructors the opportunity to contribute to the program at large through a formal process of documenting new

modular components that can be used in addition to or to replace current components. This process relies heavily on the assessment suite as a way to verify the educational value of proposed components and therefore necessitates that the assessment suite is used not only for single component evaluation but for a standard in comparing components.

This NASA Robotics Alliance Cadets Program is named an alliance as it does more than just bringing together the skills and resources of government agencies like NASA and higher level academic institutions such as Cornell University. This program also aims to incorporate the experience and support of industry and professional organizations. There has been well documented evidence that many companies strongly believe that graduating college students lack many of the key skills necessary for them to succeed in the workplace [6.10],[6.18]-[6.20] . This position was perhaps best brought out most recently in the 2006 higher education report *A Test of Leadership: Charting the Future of U.S. Higher Education* which states "Employers report that many new graduates they hire are not prepared to work, lacking the critical thinking, writing and problem-solving skills needed in today's workplaces." [6.18]

The role of industry's and professional organizations' support is not merely financial, but as the program is developed, NASA Robotics Alliance members can be asked to provide reviews on or concepts for various course components. Aside from the altruistic benefit of aiding the education field, the benefit in return for these members is a unique and potentially highly widespread promotional opportunity. Also for those groups whose products are applicable and can be donated or offered through special discounts, there is the opportunity to build their market by making their

products more familiar and relied upon by Alliance students. However, the most important target benefit is having access to significantly better potential employees and professional members.

Potential expansion into additional disciplines and higher level course development is certainly a possible extension of this project. Likewise, there is also great opportunity to spread the program down into secondary schools, potentially allowing high school students the chance to earn transferable college credit through methods already in development at Cornell. Success of the project at this stage, however, is defined as the creation of at least two courses for each of the three areas: Mechanical Engineering, Electrical Engineering, and Computer Science. These courses are significantly integrated and build upon one another's content while utilizing the robotics platform above and discussed in Section 6.4, as well as and the assessment suite discussed further in Section 6.3.

These courses will cover at least the accreditation requirements of the first two years of current courses in these three areas, and will then be evaluated using the Learning Objectives educational model and the other assessment methods mentioned above. The results of this evaluation will then be published and released to the public. Based upon the highly anticipated success of the NASA Cadets Program, the developed curriculum will be made available via the NASA Robotics Curriculum Clearinghouse as well as through limited but direct contact with schools and universities, particularly to those of significant Underrepresented / Underserved student populations. Continued support by NASA Robotics Alliance members is highly encouraged and as is mentioned above is potentially very rewarding for all

those involved. For more information on the NASA Cadets Program, please contact co-founders David Schneider or Mark León.

6.3 In Class Assessment Through Active Learning

The key to verifying that the NASA Cadets Program's goals are being met is through the development of a variety of assessment methods that can be used to establish the program's benefit to students, faculty and potential employers; to validate the credibility of the educational methods employed; and to provide a means of comparison with current and additional future methods. This section provides an overview of how the NASA Cadets Program uses Active Learning techniques to provide in-class feedback for both instructors and students while creating a positive impact on student learning. This section also provides an overview of how the Active Learning techniques themselves are assessed through the use of Learning Objective rubrics and how these rubrics are in turn used to establish the assessment suite as a standard. Section 6.4 will then use this discussion as a foundation to describe how these rubrics are used in the program to ensure accessibility and reproducible results.

The prevalent incorporation of Active Learning within the NASA Cadets Program Lesson Plans helps to enforce the value in using assessment tools not only for post-reviews of a program but for providing useful indicators to the current progress of a class. T.A. Angelo perhaps states it most succinctly as "Classroom Assessment is a simple method faculty can use to collect feedback, early and often, on how well their students are learning what they are being taught. The purpose of classroom assessment is to provide faculty and students with information and insights needed to improve teaching effectiveness and learning quality." [6.21] This view is shared by the NASA

Cadets Program. Indeed for the proven capabilities of their methods, the book *Classroom Assessment Techniques* by Angelo and Cross is identified as one of the major sources for developing the Active Learning components of the assessment suite [6.13],[6.22].

One of the aspects that is most attractive in using the methods of *Classroom Assessment Techniques* (CATs) is the seamless nature by which they can be integrated into lesson plans while jointly improving the learning experience. This view is already well supported as Schwarm and VanDeGrift noted this in 2002: "By using CATs, instructors can monitor students' learning while engaging students in reflective evaluation of course concepts." [6.22]

Many of the CATs tools, which also include active learning and self-assessment techniques, have been shown to encourage critical thinking skills in students [6.21],[6.23]. In fact, many of the NASA Cadets Program's Active Learning methods include a student self-assessment component as an integral way of building skills that are important to engineering education, such as problem-solving and lifelong learning skills [6.21],[6.24],[6.25]. Active Learning techniques have also been shown to assess and improve student learning in such targeted areas as innovation and troubleshooting [6.19],[6.20]. [1] The incorporation of these skills is particularly important as has been well voiced in a numerous educational reports such as Ref. [6.26], which states, "As is the case for many professionals, graduates of engineering education need strong critical thinking skills in a fast-changing world of increasing complexity. Critical thinking skills can be applied in professional and personal life,

[1] Although the areas of innovation and troubleshooting are relatively new, many of the concepts that these areas encompass are often grouped in the better known, better analyzed areas of problem-solving or critical thinking skills.

and are especially important to engineering education and engineers in solving problems, and designing products systems, or processes."

The variety of Active Learning exercises that can provide these multiple benefits is also substantial and hence the Lesson Plans repeatedly vary the method employed to provide presentation diversity to meet different learning styles and increase class attention. Some researchers have commented that this allows an instructor to vary the stimulus enough, much in the same way a movie special effect artist varies their tricks so that the audience accepts the method as a part of the larger presentation then recognizing its as merely an attempt to win them over. Nevertheless, the NASA Cadets Program, CAT and others have identified numerous techniques as having a particular strength in assessing knowledge of core concepts and design. These include knowledge gain tests (knowledge probes), various misconception/preconception checks, the muddiest point method, in-class or online minute papers, punctuated lectures, process analysis and analysis of performance exercises as well as CAT's Methods that are intended for use assessing lab activities and problem solving skills to name a few. [6.13],[6.22],[6.27]-[6.30] A more detailed description of the NASA Cadets Program's assessment strategies, particularly with regard to critical thinking, teamwork, communication and learning skills, can be found in Ref.[6.31].

For the purposes of this paper, this section highlights the use of the Active Learning "polling exercise" to demonstrate how the exercise's implementation is designed, how the method's assessment benefits are matched to desired Learning Objectives, how the method itself is assessed, and how the procedure is documented to allow other institutions to effectively reproduce the results.

The process begins by establishing Learning Objectives (as mentioned in Section 6.2) and then matching these Learning Objectives with an effective teaching strategy such as polling. Polling, also known as a finger signals or clickers exercise [6.32], consists of the instructor providing the class with a multiple-choice question and in response students or groups of students raise an appropriate index card or click a button from a wireless device to indicate to the instructor their own separate answer. The students' answers are visible only to the instructor, but the instructor can visually or electronically confirm that each student has answered. Some versions also allow the instructor to record student responses as a history of individual performance or at least a general distribution of answers across a class.

One of the largest benefits of this teaching strategy is that all students are forced to think about the problem and commit to an answer as the instructor is easily able to confirm answers from all students. The time required for reaching every student is equivalent to the traditional method of having a single student voice their answer. However, because they all commit to an answer, they all receive personal feedback on their performance. This in turn offers every student either validation in having achieved some level of mastery of the subject or it has forced them to realize that this area may be a source of confusion and hence they will need to focus further on or ask their own follow-up question on.

In addition, this Active Learning exercise provides a lower pressure environment for the students as only the instructor and not their peers are aware of their specific answer. This also provides the instructor with feedback as to the entire class' understanding as a whole and should a significant percentage of the class

provide the same wrong answer this offers the instructor a chance to respond to this trend immediately before attempting to build upon this material. For these reasons, this Active Learning activity is excellent to match with Learning Objectives that have been identified as commonly being associated with misconceptions. As Ref. [6.32] states "Although multiple-choice questions may seem limiting, they can be surprisingly good at generating the desired student engagement and guiding student thinking"

Variations on this activity include having groups of students offer a single answer. This in turn creates discussion among students and requires students to critique each others ideas and develop conflict resolution skills in trying to achieve a consensus. There is also great opportunity for discussion afterward. If the instructor also shares the distribution of class answers with the students, particularly when a large portion of the students answered incorrectly, the students will be more comfortable asking clarification questions, as they can see that others also had uncertainty on this point. This kind of exercise can also be repeated before and after an instruction section of the class in order to provide an even stronger measure of the effectiveness of the instruction section as well as to hopefully help students realize that learning has indeed occurred.

6.3.1 Learning Objective Rubrics

Once the entire instruction and the Active Learning exercise are complete it is crucial to perform a separate assessment of their effectiveness as well. Despite the number and validity of the methods already in existence, a significant 2006 report [6.33] still called for the need of an assessment suite that could be used as a standard of comparison by stating "These standards also should establish some requirements for

valid and reliable assessments so that accrediting organizations can provide the public some assurance that students receiving degrees or other types of credentials have the skills that institutions and programs claim." [6.33] This report is not alone however as Ref. [6.10]-[6.12],[6.34] state similar requests of the community calling for the development of "...a structured, documented system for continuous improvement." [6.12] in which comparison assessment methods can also be used to show developmental progress.

The cornerstone of creating such a standard assessment suite within the NASA Cadets Program is the development of Learning Objective rubrics. These rubrics are designed to be quick to implement and conduct and are independent of the students' specific assignments or activities. Hence they can be applied as an assessment tool for any exercise that targets the same individual abilities that the students are expected to master.

For each learning objective or desired learning outcome identified within a course, an individual rubric is constructed that is separate from grade evaluations. While an assignment or activity may touch upon many different concepts, and hence many different learning objectives, and a grade would summarize a student's mastery of these concepts combined, the rubrics summarize the students' mastery of a particular learning objective and show trends across several assignments or activities. Correlations between rubric scores and traditional course grades are typically strong, however the rubrics help to separate out which concepts a student may be struggling with or that the entire educational program is particularly effective in achieving. These rubrics also address what are traditionally deemed in engineering as "softer skills"

such as the application of communication, teamwork and problem solving skills during the assignment [6.19],[6.20].

The Learning Objective rubrics are incorporated directly into the Instructor's Manual to help ensure their proper use. The rubrics are also meant to be shared with the students, both before and after the instruction to provide students with weighted criteria for assignments and the aforementioned softer skills. In this way, the rubrics provide the entire class with a clear outline of the learning objectives for each part of the classes and the assignments. This also aids the instructors in tying the assignments to the concepts being taught in class while providing students with descriptions of the expected skill levels. The same Learning Objective rubrics may appear in several different assignments or class sessions enabling students and instructors to observe their progress throughout the semester.

Although in this section only one sample Active Learning method has been discussed, it demonstrates how these methods can be incorporated in the lesson plans to provide in-class assessment and to assess the methods themselves. Because the process is formalized and quick to implement, it is also easy to sustain or adapt to changing needs. A report on all of the assessment methods being analyzed collectively by such processes as those outlined in *Assessing Student Performance On EC2000 Criterion.* [6.35] will be made available by NASA thru the RCC with special attention given to the methods' consistency and ease of use by faculty. This assessment suite will ultimately provide the key mechanism for enabling other institutions to validate and submit their own program modules as official components to the NASA Cadets Program. Thus, the program will be able to not only incorporate the ingenuity of fellow educators, but also ensure its own continual growth and longevity.

6.4. Program Accessibility: The Robotics Platform

Active Learning tools can also be highly effective when extended to labs and more involved design projects. However the requirement to make the program as accessible and sustainable as possible, with widely realistic implementation procedures and equipment needs, can create substantial challenges. For this reason, the NASA Cadets Program heavily supports the incorporation of the low cost, highly modular robotic platform being developed by the DAVANNE LLC. Every robotic tool in the platform is designed to meet as many learning objectives as possible using as few specialized equipment pieces as possible. Hence considerable effort is spent on flexibility in the tools to allow quicker adaptation and faster learning curves for using these materials in other institutions courses as well as the NASA Cadets Lesson Plans.

As stated in Section 6.2, working with robotic systems will require students to gain a proficiency in integrating the three target areas of Mechanical Engineering, Electrical Engineering and Computer Science. More than this, using robots in an educational environment has been shown to help develop the program targets skills as identified in Sections 6.2 and 6.3. Many research studies have demonstrated the immediate value of robots as tools for students in engineering courses to relate classroom theory to its applications, and to develop their skills in problem solving and critical thinking [6.19],[6.20],[6.36]-[6.44]. Furthermore, research has also shown the long-term benefits, such as that "lessons learned (from working with robots) are not transient, and that comfort with technology and a willingness to participate in technology-related projects may be the key long-term benefits of such an educational robotics program." [6.36]

An investigation made by the NASA Cadets Program in the Fall of 2005 found that nearly all current robotic educational systems are designed for a specific task or at best a small specific set of learning objectives. However, one of the best current systems used in higher education is the Oregon State TekBot. This system was developed to focus primarily on elements of the Electrical Engineering field, but in its five year history of being used in a higher education environment, it was demonstrated that robots like the TekBot can be used to reach a wide range of learning outcomes. As is stated in Ref. [6.19], "The integration of TekBots into two freshman/sophomore courses at OSU improved several important key attributes of the course, including innovation, community, troubleshooting, depth, breadth, and professionalism." where trouble-shooting, community, and innovation are characteristics that were identified from a widely ranged survey of successful industry and academic faculty leaders as crucial components of engineering education that are not adequately targeted for today's workplace needs [6.20]. Although no current robotics system has been found to be adequate for the NASA Cadets Program cross-discipline educational needs, as, robotics studies like those performed with the TekBot provide strong evidence that the NASA Cadets Program's target skill sets can be addressed using robotics. Furthermore, these studies can also be drawn upon for existing educational robotic assessment tools that already have a proven record.

In order to establish the DAVANNE robotic system within the NASA Cadets Program as a standard across academic communities, the DAVANNE robot has been designed as a far more flexible, robust and affordable platform than previous educational robots, with enough pre-packaged features that an incoming freshman can modify significant components. At the same time, it also has been designed to incorporate enough capabilities to be scheduled for use by several cutting-edge NASA

research groups. The highly significant time and effort required for developing any robotic system for even a single task, let alone a system capable of being able to meet the educational needs of undergraduates across three disciplines as well as the needs of a NASA research scientist is substantial. However, since the potential for such a system to the educational and research community is so great, this is why NASA has taken the lead in conjunction with the DAVANNE LLC to design a robotic platform to meet this challenge [6.31] Technical specifics on the DAVANNE robot will be made available via the Robotics Curriculum Clearinghouse pending IP release, however more details on how the platform is utilized in an Active Learning setting is provided in Section 6.5.

Active Learning techniques are particularly well suited for helping to bring out the numerous psychological benefits of working with the robotics platform as well. One of the most obvious is the simple allure of being able to "own your own robot". This general appeal tied in with the stimulating creative aspects of robotic design & development captivates students' curiosity. Overall, the use of robotics as an attractive element to students is actually a very significant asset to the program. When attempting to combat the trends of attrition, the ever changing nature of a robotic platform, particularly since students are often the cause of the change, is a very useful tool for providing continual motivation and excitement.

The robot is also used to establish a sense of ownership in a project, a sense of accomplishment as robotics platforms are naturally results-oriented, as well as a sense of pride in seeing tangible results from one's labor. The NASA Cadets Program lesson plans are designed to work with the robotic platform to ensure that students experience these factors early on with Active Learning techniques used to provide quick in-class

assessment. Ultimately, success breeds success and the robotics' modular nature and packaged exercises allow students the chance to experiment and have the experience that they can indeed demonstrate a level of mastery over the material. Realization of the ability to gain proficiency in a subject matter as well as recognizing what the proficiency of skills enables them to accomplish, are highly empowering events for students. Furthermore it is events like this that encourage them to look for the value in lessons on their own and to even reach out for knowledge outside of the standard curriculum [6.45]. As is stated in Ref. [6.36], the "…positive impact of (robotics) on student learning (extends) well beyond the boundaries of specific technical concepts in robotics" Hence it is through experiences like those provided through incorporating robotics that the ability to innovate is born.

To aid in the development of this ability, NASA has traditionally encouraged the formation of nationwide competitions, the most famous of which is the US FIRST robotics competition which was supported in part by Apollo XI Astronaut Buzz Aldrin. Today this program has spread to over 800 high schools across the U.S. [6.46] Competitions offer a mixture of well specified goals, with constrained problems and yet leave open areas for invention and experimentation. Thereby competitions can offer more controlled and even more learning outcome targeted versions of real world scenarios. After all, it is now common knowledge that "the development of any skill is best facilitated by giving students practice and not by simply talking about or demonstrating what to do." [6.47] More than providing a link from theory to practice, the process of dealing with the competition's challenges and constraints while attempting creative solutions inevitably force students to gain experience in troubleshooting. In addition competitions also generate an incentive for students to

excel and “win” that can often exceed the drive created by offering only grading rewards for achievement.

For this reason, the NASA Cadets Program is developing several competitions that range from laboratory experiments and weekly homework “challenge problems” to year-long projects. Many of NASA’s current competitions will provide inspiration for these new competitions, as will competitions outside of NASA, such as the worldwide RoboCup Competition, in which program contributor Cornell University has been world champion 4 out of the 7 times it competed.

The key in developing the competitions is that the rules and execution of the competition is constructive to the student community. This can be achieved by tapering the emphasis on “winning” as compared to promoting every student and team to simply “score” the best that they can. Allowing students various areas to succeed can aid in creating this environment and simultaneously help create diversity in students’ solutions. Once again the use of rubrics and their explanation and open availability to students becomes a useful tool. With the design of multiple success criteria into a competition, this also creates a need for students to prioritize goals, budget resources and ultimately develop project management skills. Furthermore adding to the experience Active Learning exercises like those mentioned in Section 6.3, allows instructors to highlight pitfalls, ensure students are taking into consideration all the requirements and constraints, as well as be a conduit of general discussion on these design concepts.

Similarly, in any situation where multiple solutions are possible, the need for effective communication for describing the reasoning behind decision making

becomes self-evident. Therefore having a base system, like the robotic platform, that all students are working from encourages the exchange of ideas and a common language for passing knowledge between students. Furthermore, including elements like Active Learning that allow peer assessment through various forms of constructive criticism can also help to build community. Combining all these benefits, it becomes clear that the robotic platform will be an exceptional tool in ensuring that the NASA Cadets Program will reach its goals.

6.5 Lesson Plan, Active Learning, Robotics Platform and Assessment Suite Integration

One of the founding concepts behind the NASA Cadets Program is that the integration of the assessment suite and the robotic platform with the lesson plans will result in more effective products than any component would be on its own. To demonstrate this integration, this section outlines one standalone module of the NASA Cadets Program called the Robotics Triathlon that was originally designed for Cornell University.

As the name implies, the Robotics Triathlon is a three-part competition. The target audience for this module is incoming Freshmen with little to no experience in any of the three target areas (Mechanical Engineering, Electrical Engineering, and Computer Science). The time frame for this module is two 2- hour lab sessions with a 2-week period in between each lab. The class size is approximately 30-40 students, broken into groups of 3-4. The equipment provided to each group is one PC station and a single robot with a set of modular components, along with handouts and a small 15-page C++ reference guide, which will be described later in this section. The

recommended instructor support is one key lecturer and 1-2 teaching assistants who are familiar with the equipment.

The main learning goals of the module for the three target areas can be described most easily by walking through the implementation of the Robotic Triathlon. This description is intentionally made general in parts in order to convey to the reader more of an overview of the style of the NASA Cadets Programs deliverables. The module actually begins about 1-2 weeks before the actual first lab, i.e. perhaps in an earlier laboratory session or classroom lecture. In this session, the instructor lays out the Robotic Triathlon Competition as well as communicates the precise learning objectives for the students for the Robotic Triathlon lab sessions. Furthermore, the instructor also issues the first part of a knowledge gain exam on the learning objectives.

After completion of the exam, the students are then given a copy of a small C++ reference guide. The reference guide covers the topics of a few variable types, arithmetic, relational and logical operators, as well as if/else statements and while loops in 15 pages. Students are asked to review the reference guide and complete 2 pages of worksheets before the first lab. The students are also asked to complete a third brief worksheet the night just before their first lab session to allow the concepts to be fresh in their minds. The anticipated time required for the students to complete these tasks is approximately 3 hours and the students' worksheets are collected at the beginning of the first lab session.

In the first lab session, the students are engaged in active learning using such techniques as polling to review the material read, address any misconceptions and to

be introduced to a compiler. Through a step by step process the students slowly build a program to give them experience with the material they learned as they work towards programming the robot to move forwards and backwards and turn to either side by responding to keystrokes from the PC keyboard. As was introduced in Section 6.4, in order to make this project feasible for incoming Freshmen, pre-packaged components such as low level motor control, communication protocols and other platform functionality is already provided for the students and these components' use is simplified with the aid of wrapper functions.

Aside from merely practicing the material, throughout this lab session students are challenged via Active Learning methods, like those mentioned in Section 6.3, to identify errors in given code and assess for themselves what the outcome of various code changes may be. This in turn helps to target the higher levels of Bloom's Taxonomy as well as the key area of troubleshooting. Also as certain students have difficulties with various components during the lab, these issues are addressed in such a manner that a student is not dubbed completely wrong but rather the situation is that "one of your fellow student teammates needs the class's help". This can obviously help bring attention to typical mistakes to the entire class, but potentially even more importantly this can be used to instil the sense of community and the need for teamwork. As small syntax errors are both common and often relatively easy to correct with programs of this scale, more than just reinforcing troubleshooting skills, this introduces early on a relatively safe environment for students to make mistakes in. Furthermore, as the negative impact of making a mistake is minimal, this can actually reduce the fear of failure and increase the willingness to experiment and readiness to innovate in the next lab section. The students are challenged at the end of the first lab

session to modify their code in order to have the robot drive in a square with only a "Go" input from the keyboard.

The session ends with the use of assessment methods mentioned in Section 6.3.1 in order to determine how effective the lesson was and to provide students post-session feedback on their abilities as well. The instructor also provides an introduction for the students on the next homework and lab section with a particular focus on how these activities relate to the top three levels of Bloom's Taxonomy: Analysis, Synthesis and Evaluation.

In the homework assignment for the next lab session, the students are given a problem where they must choose a limited set of vectors from a provided library of potential vectors that can be combined to transverse several simple maze-like grids. At face value the problem provides an introduction to the concepts of algorithm development, but the solution reporting process is geared to ultimately force students to first formally analyze the problem's constraints and requirements. Then students must develop their solutions and evaluate them themselves based upon provided criteria in the same fashion as the Active Learning troubleshooting exercise they experienced during the first lab session. The familiarity of the exercise aids the student in realizing the benefit even though they are now asked to perform the same activities on their own.

In the last step of the homework assignment, the experience is taken further by allowing students to modify one of the constraints and provide reasoning on why this relaxation would allow potential solutions that would better meet the problem's requirement criteria. Finally, students are made aware once again that the process they

just followed fits within the Analysis, Synthesis and Evaluation levels of Bloom's Taxonomy. It is important to note, however, that if the students' curriculum has not yet covered vectors and vector addition, a suggested lesson plan is provided as a part of this module.

The second lab session begins with a more specific description of the Robotic Triathlon. In the Robotic Triathlon each team of students will be asked to modify their robot to increase its ability to navigate an obstacle course and perform some timed simple tasks. To prepare the students for this task, students are then led through a small series of active learning individual exercises to teach the Mechanical Engineering concepts of gear ratios and torque. Students are also given a very general overview of the ideas of feedback control and the incorporation of sensors from more of an Electrical Engineering perspective, which will also be useful knowledge for them in making modification decisions for their robots.

This instructional component is designed to last no more than 45 minutes allowing the students 1 hour and 15 minutes to make the modifications. However during this instruction, several Active Learning exercises are conducted so that the students have a better awareness of their own personal capabilities and what they will be able to offer to the design group and what areas they may want to confer with others or the instructor before moving forward. The instructor is also able to identify whether large groups of the class are having troubles with a particular area and hence address the issue with the entire class then instead of having to repeat the clarification to each student group during the Triathlon.

The modifications the students are allowed to make are (1) changing the gearing of the robot's motors using provided gears, (2) changing the length of an arm of a provided gripper tool on the robot, i.e. which has influence on the torque the arm can provide, and (3) modifying a gain input to a provided function that influences the robot's motion controls where there are trade-offs such as between speed and control sensitivity. Due to the modular nature of the robotics platform, all of these changes can be done within a few minutes time, allowing the students significant time to consider their design choices carefully. Once the group has made their modifications, the students run their robot through the course and receive a score based upon their task performance and completion time.

Each student group is actually allowed to run their robot through the Triathlon twice. After receiving the score for their first run, students are allowed to make any changes to their robot once again and then run the robot for a second time. The best of their two runs' scores is the group's final score. However, the score itself counts for only a small amount of the students grade and far more weight is given to the calculations and reasoning used to justify their modifications.

The second lab session as described here clearly demonstrates how many of the NASA Cadets Program's targeted areas can be integrated together. Topics in all three disciples of Mechanical Engineering, Electrical Engineering and Computer Science are covered simultaneously. Similarly, the students are asked to innovatively use of the provided components to meet the challenges of the Triathlon. The implementation of their modifications and multi-run aspect of the Triathlon will give experience in troubleshooting. Then all throughout the event the group set-up and

competition component of the module aid in the development of the community target area.

The community target area as well as other elements of the module are also ameliorated through the use of assessment suite components throughout the module's execution. Peer review and constructive criticism exercises are also used as a component of the module's assessment. Additionally, throughout the module, students are asked to employ self-assessment techniques to both aid in their design process and in the instructors evaluation of the module's execution.

The students' final reports include both team submission and individual submission components to not only ensure both group and individual accountability, but as an evaluative check to the in-class assessment components. The questions the students are asked to address in these reports also delve into the Analysis, Synthesis and Evaluation levels of Bloom's Taxonomy as well as the innovation, troubleshooting, and community target areas. By measuring the students responses using the verified rubrics mentioned in Section 6.3, the report can also aid in the module assessment. Furthermore the report is also used as an assessment tool by making part of the report's individual component the second half of the knowledge gain test. Indirect measures such as surveys and interviews can also be employed for additional data collection.

As a final step to the module, the instructor is encouraged to share and discuss the results of all of the evaluation tools with the students as a group, while reminding students that their grades are independent of the assessment tools results. This can help to both reiterate to the students the value of each component of the module and

especially the assessment methods employed as well as aid students in being able to identify the value of future module's components on their own.

6.6 Conclusions

The NASA Robotics Alliance Cadets Program's assessment suite is focused on fulfilling the nationally recognized need for a standard system to identify the most effective innovations within today's engineering education programs. Key components to this suite that have been recognized by ASEE are the Active Learning activities which have been extended and enhanced by the NASA Cadets / DAVANNE robotics platform. Together these tools achieve the accessibility and sustainability needs as well as the program validation requirements established by the 2006 *National Defense Education and Innovation Initiative* report. Additional educational benefits of the program to such target areas as troubleshooting, innovation, and community are also highlighted in the description of the NASA Cadet Robotics Triathlon module developed in part with Cornell University.

REFERENCE

[1] Benjamin S. Bloom, "Taxonomy of Educational Objectives", *Published by Allyn and Bacon, Boston, MA.* Copyright (c) 1984

[2] Al-Bahi. A.M., "Development of a Design Phase Checklist for Outcome Based Active/Cooperative Learning" *Conferences &Exhibition of the American Society of Engineering Education*, June 2006

[3] Johnson, D.W., R.T. Johnson and K.A. Smith "Active Learning: Cooperation in College Classroom" *2nd ed. Edina, MN:Interavtive Bppk Co.* 1998

[4] O.Farook, C.Sekhar, J.Agrawal, E. Bouktache, A.Ahmed, M.Zahree,"Outcome Based Education and Assessment", *Conferences &Exhibition of the American Society of Engineering Education*, June 2006

[5] Y. Dori and J. Belcher "Effect of Visualizations and Active Learning on Students' Understanding of Electromagnitism Concepts" *National Association for Research in Science Teaching (NARST) Proceedings*, 2003.

[6] J.Linsey, B.Cobb, D.Jensen, K.Wood, S.Eways, "Methodologies and Tools for Developing Hands-On Active Learning Activities" *Conferences &Exhibition of the American Society of Engineering Education*, June 2006

[7] M.Prince and M.Vigeant "Using Inquery-Based Activities to Promot Understanding of Critical Engineering Concepts" *Conferences &Exhibition of the American Society of Engineering Education*, June 2006

[8] Felder, R.M., and R. Brent "Designing and Teaching Courses to Satisfy the ABET Engineering Criteria" *Journal of Engineering Education*, 92 (1), 7-25, 2003.

[9] "NASA Education Enterprise Strategy", *National Aeronautics and Space Administration*, 2003

[10] "National Defense Education and Innovation Initiative: Meeting America's Economic and Security Challenges in the 21st Century", *Report from the Association of American Universities*, January 2006

[11] Lattuca, Terenzini, Volkwein & Peterson, "Engineering Change: A Study of the Impact of EC2000", *ABET*, 2006.

[12] "NSF 95-65 Restructuring Engineering Education: A Focus on Change", *Report of a National Science Foundation Workshop on Engineering Education*, April 1995

[13] Angelo, T. A. and Cross, K. P., "Classroom Assessment Techniques, A Handbook for College Teachers", *2nd ed., Jossey-Bass Publishers, San Francisco*, 1993

[14] Angelo, T.A., "Ten easy pieces: Assessing higher learning in four dimensions", *In Classroom Research: Early lessons from success. New directions in teaching and learning*, (#46), Summer, 17-31, 1991

[15] D. Schneider, "*NASA Robotics Alliance Cadets Program White Paper*", September 2005

[16] Frank Huband, *Conversation* 3/27/2007

[17] "Sustaining the Change: A Follow-Up Report to the Vision for Change"*, ABET* 2006

[18] *"*A Test of Leadership: Charting the Future of U.S. Higher Education", *US. Department of Education*, 2006

[19] Heer, D. Traylor,R.L., Thompson, T., Fiez, T., "Enhancing the Freshman and Sophomore ECE Student Experience Using a Platform for Learning.™", *IEEE Transactions On Education*, Vol. 46, No. 4, November 2003.

[20] Heer, D., Traylor, R. Fiez, Terri, "Tekbots™: Creating Excitement for Engineering Through Community, Innovation and Troubleshooting", *Frontiers in Education Conference*, 2002

[21] Angelo, T. , "Classroom assessment: Guidelines for success", Teaching excellence: Toward the best in the academy, 12 (4), 1-2. North Miami Beach, FL: *The Professional and Organizational Development Network in Higher Education Essays on Teaching Excellence series*. 2000

[22] Schwarm, S., VanDeGrift, T. "Using Classroom Assessment to Detect Students' Misunderstanding and Promote Metacognitive Thinking", *Keeping Learning Complex: The Proceedings of the Fifth International Conference of the Learning Sciences*, 2002

[23] Soetaert, E., "Quality in the classroom: Classroom assessment techniques as TQM", In T. Angelo (Ed.), Classroom assessment and research: Uses, approaches, and research findings (pp. 47-55). *New Directions for Teaching and Learning*, no. 75. San Francisco: Jossey-Bass, 1998

[24] Ornstein, A.C. and Hunkins, F.P., "Curriculum Foundations, Principles, and Issues", *3rd ed., Needham Heights, MA: Allyn & Bacon*, 1998

[25] Posner, G. J., & Rudnitsky, A. H. , "Course Design: A Guide to Curriculum Development for Teachers"*, 6th ed., New York: Addison Wesley Longman, Inc.*, 2001

[26] Ceylan, T., Wah Lee, L. "Critical Thinking And Engineering Education" *Paper presented at the Sectional Conference, American Society for Engineering Education*, April, 2003.

[27] Brinkman, G.; van der Geest, T., "Assessment of Communication Competencies in Engineering Design Projects" *Technical Communication Quarterly, 12, 1; ABI/INFORM Global*, pg. 67. Winter 2003

[28] Olds, B.M., Moskal, B.M., and Miller, R.L., "Assessment in Engineering Education: Evolution, Approaches, and Future Collaborations", *Journal of Engineering Education*, Vol. 94, No. 1, 2005.

[29] Kaufman, D., Felder, F., "Peer Ratings in Cooperative Learning Teams", *Proceedings of the 1999 Annual ASEE Meeting, ASEE*, June 1999.

[30] Erwin, T. Dary, "The NPEC Sourcebook on Assessment, Volume 1: Definitions and Assessment Methods for Critical Thinking, Problem Solving, and Writing", Prepared for the National Postsecondary Education Cooperative and its Student Outcomes Pilot Working Group under the sponsorship of the National Center for Education Statistics, *U.S. Department of Education*. 2000.

[31] D.Schneider, C van der Blink, "An Introduction to the NASA Robotics Alliance Cadets Program" *Proceedings of the ASEE St. Lawrence Division I3E2 Conference*, Nov. 2006

[32] C.Wieman and K.Perkins "Transforming Physics Education" *Physics Today*, Nov.2005

[33] Schray, V., "Assuring Quality in Higher Education: Recommendations for Improving Accreditation", Issue paper. *U.S. Department of Education*. 2006

[34] "Australian Department of Education, Science and Training, Status and Quality of Teaching and Learning of Science in Australian Schools", *Commonwealth of Australia*, 2005

[35] Strauss, L., Terenzini, P., "Assessing Student Performance On EC2000 Criteria", *Proceedings of the 2005 American Society for Engineering Education Annual Conference & Exposition*, 2005

[36] Nourbakhsh, I., Crowley, K., Bhave, A. , Hamner, E., Hsiu, T. "The Robotic Autonomy Mobile Robotics Course: Robot Design, Design and Educational Assessment", *Autonomous Robots 18*, 103–127, 2005

[37] Thompson, T., Heer, D., Brown, S., Traylor, R.L., Fiez, T. "Educational Design, Evaluation, & Development of Platforms for Learning", *ASEE/IEEE Frontiers in Education Conference*, 2004

[38] Manseur, R., "Development of an Undergraduate Robotics Course", *Frontiers in Education Conference*, pp. 610-612, 1997

[39] Mehrl, D.J., Parten, M.E., Vines, D.L., "Robots Enhance Engineering Education", *Frontiers in Education Conference*, pp. 613-618, 1997

[40] Pomalaza-Ráez, C., Groff, B., "Retention 101: Where Robots Go…Students Follow", *Journal of Engineering Education*, January 2003

[41] Kumar, A.N. "Using Robots in an Undergraduate Artificial Intelligence Course: An Experience Report", *In Proceedings of the 31st Frontiers in Education Conference*, Vol. 2, 2001

[42] Chamilothoris, G., Papoutsidakis, M.,Shaping "The Mechatronics Course for the Control Curriculum", *International Federation of Automatic Control*, July 2005

[43] Greene, J.R., "Experience-based learning in electrical and electronic engineering: An effective and affordable approach", *AFRICON*, September 1992

[44] Pierre, J.S., Christian, J., "K-12 initiatives: increasing the pool", *Frontiers in Education*, 2002.

[45] "Motivating Students for Lifelong Learning", *Organization for Economic Co-Operation and Development (OCED)*, 2000

[46] "*FIRST Robotics Competition*" , http://www.usfirst.org/robotics/index.html, accessed 10/12/06

[47] Woods, D.R., Felder, R., Rugarcia, A., Stice, J., "The Future of Engineering Education III. Developing Critical Skills", *Chemical Engineering Education*, 34(2), 108-117, 2000

CONCLUSIONS

This dissertation offers the following conclusions and contributions:

- RoboFlag, versions 1.1, 2.0, 2.1 and 3.0 was developed as a high fidelity robotic testbed and simulation system under the supervision of the author as Program Manager.

- The G^*_{TA} method was created as an optimal task allocation method which is capable of producing optimal solutions to the NP-Hard task allocation method in computational runtimes that are on average more than an two orders of magnitude faster than a standard Mixed Integer Linear Programming (MILP) method.

- The A^*MILP and G^*MILP methods were created as a combination of the G^*_{TA} method and MILP techniques. These methods are capable of producing optimal solutions to the NP-Hard task allocation method in computational runtimes that are on average more than an order of magnitude faster than a standard Mixed Integer Linear Programming (MILP) method. They have been shown to have significantly better scaling potential over G^*_{TA} or MILP methods.

- The G^*_{TA} method was established as an anytime solution capable method that for small scale problems under strict time constraints of 0.025 seconds can produce approximation solutions to NP-Hard task allocation problems on average with less than a 2% error from optimal.

- The optimistic predictive cost function of the G^*_{TA} method was improved which resulted in average computational runtimes that were 5 times faster than when using the original G^*_{TA} optimistic predictive cost function. The new optimistic predictive cost function concurrently also reduced the average memory usage by an order or magnitude over the original G^*_{TA} optimistic predictive cost function.

- A hierarchical adaptive K-means clustering technique was developed that through providing guarantees on the cluster size and the number of clusters at the top level of the hierarchy, has been shown to be a practical means for partitioning large scale NP-Hard problems, like the task allocation problem.

- The Neighbor H-G^*_{TA} approximation method was created for NP-Hard task allocation problems which for large scale problems on the order of 2-6 sources and 50-250 targets is capable of producing comparable solutions to a standard greedy method in computation times that are up to an order of magnitude faster than the standard greedy method.

- Several variations of the H-G^*_{TA} method were created that followed traditional computer science thought but ultimately verified the approach of the Neighbor H-G^*_{TA} as a fast and viable approximation method for large scale, real-time NP-Hard task allocation problems.